SAND REPORT

SAND2003-0887
Unlimited Release
Printed March 2003

Controlatron Neutron Tube Test Suite Software Manual Controlatron Interlock Control (V2.2)

William P. Noel, Debra L. Wallace, Monica Martinez
Robert Hertrich and Keith Barrett

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94-AL85000.

Approved for public release; further dissemination unlimited.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone:	(865)576-8401
Facsimile:	(865)576-5728
E-Mail:	reports@adonis.osti.gov
Online ordering:	http://www.doe.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone:	(800)553-6847
Facsimile:	(703)605-6900
E-Mail:	orders@ntis.fedworld.gov
Online order:	http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2003-0887
Unlimited Release
Printed March 2003

Controlatron Neutron Tube Test Suite Software Manual Controlatron Interlock Control (V2.2)

William P. Noel and Debra L. Wallace
Neutron Tube Design Department

Monica Martinez
Neutron Generator Design Department

Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-0516

Robert J. Hertrich and Keith Barrett
Primecore Systems, Inc.
Albuquerque, New Mexico

Abstract

The Controlatron Software Suite is a custom built application to perform automated testing of Controlatron neutron tubes. The software package was designed to allow users to design tests and to run a series of test suites on a tube. The data is output to ASCII files of a pre-defined format for data analysis and viewing with the Controlatron Data Viewer Application. This manual discusses the operation of the Controlatron Interlock Control (CIC) Module from the Controlatron Neutron Tube Test Suite of Software.

Revision History

Rev #	Author	Date	Description
1.0	William Noel	10/08/01	Original Layout and Formatting
1.2	William Noel Monica Martinez	3/16/02	Rough Draft Revision 3\Editing\Format to SNL SAND specifications
2.0	William Noel	3/20/02	Final Revision

Figure 1. Revision History Table

Robert J. Hertrich
PrimeCore Systems, Inc.
8421 Osuna Rd. NE, Suite C-1
Albuquerque, New Mexico 87111

Keith Barrett
PrimeCore Systems, Inc.
8421 Osuna Rd. NE, Suite C-1
Albuquerque, New Mexico 87111

William P. Noel, EMT-B, RN
Neutron Tube Design Department
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-0516

Debra L. Wallace
Neutron Tube Design Department
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-0516

Monica L. Martinez
Neutron Generator Design Department
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-0516

Contents

1. Controlatron Software Overview 13

1.1 Overview 13
1.2 Software Specifications 13
1.3 System Administrators 14
1.4 Hardware Considerations 14
1.5 Configuring the Interlock System 14
1.6 Message Manager Architecture Description and Tutorial 18
1.6.1 What is a state machine? 19
1.6.2 What is a 'State?' 19
1.6.3 What is the Message Manager? 20
1.7 Overall Architecture of a Message Manager Based Program 23
1.8 Remote Message Manager 24
1.9 Advantages of a State Machine in General 24
1.10 Advantages of the Message Manager 24
1.11 Disadvantages of the Message Manager 25
1.12 Object Orientation Using the Message Manager 25
1.13 Queued State Machine Architecture Description and Tutorial 26
1.13.1 First of all, what is a state machine? 27
1.13.2 What is a queued state machine? 27
1.13.3 How do I add events to the queue? 27
1.13.4 Timed Events 28
1.13.5 What can I do in a "State?" 29
1.13.6 Advantages of a State Machine 29

2. Controlatron Interlock Control System 31

2.1 CIC Top Level.vi 31
2.1.1 Connector Pane 31
2.1.2 Front Panel 31
2.1.3 Controls and Indicators 31
2.1.4 Top Level Diagram 38
2.1.5 Controlatron Interlock Control 38
2.1.6 List of SubVIs 45
2.2 Unload Module.vi 46
2.2.1 Connector Pane 46
2.2.2 Front Panel 47
2.2.3 Controls and Indicators 47
2.2.4 Block Diagram 48
2.2.5 List of SubVIs 49
2.3 Queue Manager.vi 50
2.3.1 Connector Pane 50
2.3.2 Front Panel 50
2.3.3 Controls and Indicators 50
2.3.4 Block Diagram 52
2.3.5 List of SubVIs 54

2.4 Log System Event.vi 54
2.4.1 Connector Pane 54
2.4.2 Front Panel 54
2.4.3 Controls and Indicators 55
2.4.4 Block Diagram 56
2.4.5 List of SubVIs 57
2.5 General Error Handler.vi 57
2.5.1 Connector Pane 57
2.5.2 Front Panel 58
2.5.3 Controls and Indicators 58
2.5.4 Block Diagram 61
2.5.5 List of SubVIs 64
2.6 Generate Error.vi 64
2.6.1 Connector Pane 64
2.6.2 Front Panel 64
2.6.3 Controls and Indicators 64
2.6.4 Block Diagram 66
2.6.5 List of SubVIs 67
2.7 Register Module.vi 67
2.7.1 Connector Pane 67
2.7.2 Front Panel 67
2.7.3 Controls and Indicators 68
2.7.4 Block Diagram 69
2.7.5 List of SubVIs 69
2.8 MA Definitions.vi 70
2.8.1 Connector Pane 70
2.8.2 Front Panel 70
2.8.3 Controls and Indicators 70
2.8.4 Block Diagram 71
2.8.5 List of SubVIs 71
2.9 Initialize Module Manager.vi 71
2.9.1 Connector Pane 71
2.9.2 Front Panel 71
2.9.3 Controls and Indicators 71
2.9.4 Block Diagram 72
2.9.5 List of SubVIs 73
2.10 Set Windows State.vi 73
2.10.1 Connector Pane 73
2.10.2 Front Panel 74
2.10.3 Controls and Indicators 74
2.10.4 Block Diagram 76
2.10.5 List of SubVIs 77
2.11 Windows State.ctl 78
2.11.1 Connector Pane 78
2.11.2 Front Panel 78

2.11.3 Controls and Indicators ... 78
2.11.4 Block Diagram ... 79
2.11.5 List of SubVIs ... 79
2.12 Send Message-Get Response.vi ... 79
2.12.1 Connector Pane ... 79
2.12.2 Front Panel ... 79
2.12.3 Controls and Indicators ... 79
2.12.4 Block Diagram ... 81
2.12.5 List of SubVIs ... 82
2.13 Return Reference Manager.vi ... 82
2.13.1 Connector Pane ... 82
2.13.2 Front Panel ... 82
2.13.3 Controls and Indicators ... 82
2.13.4 Block Diagram ... 84
2.13.5 List of SubVIs ... 86
2.14 create gevent.vi ... 86
2.14.1 Front Panel ... 87
2.14.2 Controls and Indicators ... 87
2.14.3 Block Diagram ... 88
2.14.4 List of SubVIs ... 89
2.15 jLibrarian.vi ... 89
2.15.1 Connector Pane ... 89
2.15.2 Front Panel ... 90
2.15.3 Controls and Indicators ... 90
2.15.4 List of SubVIs ... 93
2.16 Librarian File Info In.ctl ... 93
2.16.1 Connector Pane ... 93
2.16.2 Front Panel ... 94
2.16.3 Controls and Indicators ... 94
2.16.4 List of SubVIs ... 95
2.17 Librarian File Info Out.ctl ... 95
2.17.1 Connector Pane ... 95
2.17.2 Front Panel ... 95
2.17.3 Controls and Indicators ... 95
2.17.4 List of SubVIs ... 96
2.18 Librarian File List.ctl ... 96
2.18.1 Connector Pane ... 96
2.18.2 Front Panel ... 96
2.18.3 Controls and Indicators ... 96
2.18.4 List of SubVIs ... 97
2.19 Destroy gevent.vi ... 97
2.19.1 Connector Pane ... 97
2.19.2 Front Panel ... 98
2.19.3 Controls and Indicators ... 98
2.19.4 Block Diagram ... 99

2.19.5 List of SubVIs ... 100
2.20 Wait on gevent.vi ... 100
2.20.1 Connector Pane ... 100
2.20.2 Front Panel ... 100
2.20.3 Controls and Indicators ... 100
2.20.4 Block Diagram ... 102
2.20.5 List of SubVIs ... 103
2.21 Get Message.vi ... 103
2.21.1 Connector Pane ... 103
2.21.2 Front Panel ... 103
2.21.3 Controls and Indicators ... 103
2.21.4 Block Diagram ... 104
2.21.5 List of SubVIs ... 105
2.22 Send Message.vi ... 105
2.22.1 Connector Pane ... 106
2.22.2 Front Panel ... 106
2.22.3 Controls and Indicators ... 106
2.22.4 Block Diagram ... 107
2.22.5 List of SubVIs ... 108
2.23 CIC Set Interlock Setup Change.vi ... 108
2.23.1 Connector Pane ... 108
2.23.2 Front Panel ... 109
2.23.3 Controls and Indicators ... 109
2.23.4 Block Diagram ... 110
2.23.5 List of SubVIs ... 110
2.24 CIC Query Interlock State.vi ... 110
2.24.1 Connector Pane ... 110
2.24.2 Front Panel ... 110
2.24.3 Controls and Indicators ... 110
2.24.4 Block Diagram ... 111
2.24.5 List of SubVIs ... 111
2.25 CAS Set Interlock State.vi ... 111
2.25.1 Connector Pane ... 111
2.25.2 Front Panel ... 112
2.25.3 Controls and Indicators ... 112
2.25.4 Block Diagram ... 113
2.25.5 List of SubVIs ... 113
2.26 CCC Set Interlock State.vi ... 113
2.26.1 Connector Pane ... 113
2.26.2 Front Panel ... 113
2.26.3 Controls and Indicators ... 113
2.26.4 Block Diagram ... 114
2.26.5 List of SubVIs ... 114
2.27 PCS 01 001A Set Interlock State.vi ... 114
2.27.1 Connector Pane ... 114

2.27.2 Front Panel 115
2.27.3 Controls and Indicators 115
2.27.4 Block Diagram 116
2.27.5 List of SubVIs 116
2.28 CIC Variables.ctl 116
2.28.1 Connector Pane 116
2.28.2 Front Panel 117
2.28.3 Controls and Indicators 117
2.28.4 List of SubVIs 118
2.29 CIC Channel Setup.ctl 118
2.29.1 Connector Pane 118
2.29.2 Front Panel 118
2.29.3 Controls and Indicators 118
2.29.4 List of SubVIs 118
2.30 CIC DAQ Get Interlock Status.vi 118
2.30.1 Connector Pane 118
2.30.2 Front Panel 119
2.30.3 Controls and Indicators 119
2.30.4 Block Diagram 120
2.30.5 List of SubVIs 121
2.31 Read from Digital Port_With Error.vi 121
2.31.1 Connector Pane 121
2.31.2 Front Panel 122
2.31.3 Controls and Indicators 122
2.31.4 Block Diagram 123
2.31.5 List of SubVIs 124
2.32 DIO Port Config.vi 124
2.32.1 Connector Pane 124
2.32.2 Front Panel 124
2.32.3 Controls and Indicators 124
2.32.4 Block Diagram 126
2.32.5 List of SubVIs 126
2.33 DIO Port Read.vi 126
2.33.1 Connector Pane 127
2.33.2 Front Panel 127
2.33.3 Controls and Indicators 127
2.33.4 Block Diagram 128
2.33.5 List of SubVIs 129
2.34 Driver Demo Global.vi 129
2.34.1 Connector Pane 129
2.34.2 Front Panel 129
2.34.3 Controls and Indicators 130
2.34.4 List of SubVIs 131
2.35 Respond To Message.vi 131
2.35.1 Connector Pane 131

2.35.2 Front Panel 131
2.35.3 Controls and Indicators 131
2.35.4 Block Diagram 133
2.35.5 List of SubVIs 133
2.36 Set gevent.vi 133
2.36.1 Connector Pane 133
2.36.2 Front Panel 134
2.36.3 Controls and Indicators 134
2.36.4 Block Diagram 135
2.36.5 List of SubVIs 136
2.37 Get Response Reference.vi 136
2.37.1 Connector Pane 136
2.37.2 Front Panel 136
2.37.3 Controls and Indicators 136
2.37.4 Block Diagram 138
2.37.5 List of SubVIs 138
2.38 Simple Error Handler.vi 138
2.38.1 Connector Pane 138
2.38.2 Front Panel 138
2.38.3 Controls and Indicators 139
2.38.4 Block Diagram 140
2.38.5 List of SubVIs 140
2.39 CIC File Management.vi 140
2.39.1 Connector Pane 140
2.39.2 Front Panel 140
2.39.3 Controls and Indicators 141
2.39.4 Block Diagram 142
2.39.5 List of SubVIs 143
2.40 Get Screen Size.vi 143
2.40.1 Connector Pane 143
2.40.2 Front Panel 143
2.40.3 Controls and Indicators 144
2.40.4 Block Diagram 144
2.40.5 List of SubVIs 144

Figures

Figure 1. Revision History Table 4
Figure 2. System Administrators 14
Figure 3. (Previous Page) Controlatron Interlock Control Center Panel 15
Figure 4. Controlatron Interlock Control Module Idle Case 16
Figure 5. Controlatron Interlock Control Module 'Get Interlock State' Case 17
Figure 6. Example Queued State Machine 'Idle State' 19
Figure 7. CIC Top Level Diagram (Message Manager Idle Event) 20
Figure 8. Packing Data For Message Management 22
Figure 9. Un-Packing Data Received in a Message 22
Figure 10. Packing and Un-Packing Data Summary 22
Figure 11. General Message Manager Architecture 23
Figure 12. Module Responsibility Layout 23
Figure 13. Example 'Access Function' 26
Figure 14. Adding Events to the 'Queue' of a Queued State Machine 28
Figure 15. Programmatically adding events to the Queue 28

(INTENTIONALLY LEFT BLANK)

1. Controlatron Software Overview

1.1 Overview

The Controlatron Software Suite is a custom built application to perform automated testing of Controlatron neutron tubes. The software package is capable of allowing users to design tests and to run a series of test suites on a tube. The data is output to ASCII files of a pre-defined format for data analysis and viewing with the Controlatron Data Viewer Application.

The software is designed to specifically control the system hardware defined by the Sandia documents: Controlatron Tube (Wire Reservoir) Test Procedure, drawing #10024353-000 and Product Specification, Controlatron Tube (U), drawing #PS704791. The software only controls the hardware designed with the system, but the source code was written with a modular design that is adaptable to rapid change.

The Controlatron Software Suite is comprised of four software manuals:

1. CIC Controlatron Interlock Control
2. CAS Controlatron Acquisition System
3. CCC Controlatron Control Center
4. PCS Power Control System

These four manuals working in concert as a quad state machine provide the necessary instrument setup and control to perform a test.

This document will detail the CIC module. This manual handles all the interlock and safety aspects of the test suite.

1.2 Software Specifications

Operating System:	Windows 2000
Programming Language\Version:	LabVIEW 6.02 (LabVIEW 6i with the 6.02 update patch)
NIDAQ Version:	NIDAQ 6.9 (National Instruments)
FieldPoint Version:	Fieldpoint 2.0 (National Instruments)
Message Manager Version:	v5.0.6
GPIB Version:	NI 488.2 Version 1.70
Executable Program Name:	Controlatron Test Suite Rev*X*_*X*.exe
Top Level Program Name:	CCC Top Level.vi
Data Viewer App Name:	Controlatron Data Viewer Rev*X*_*X*.exe
Minimum System Requirement:	Pentium II, 400MHz or better PC
Memory Requirement:	128 Mb Minimum, 256 Mb recommended

Network Requirement: TC/PIP protocol network connection (for data storage)

1.3 System Administrators

Name	Phone	Email	Company
William P. Noel	505-845-8796	wpnoel@sandia.gov	SNL
Robert Hertrich	505-294-0010	rhertrich@primecoresystems.com	PrimeCore Systems, Inc.
Keith Barrett	505-294-0010	kbarrett@primecoresystems.com	PrimeCore Systems, Inc.

Figure 2. System Administrators

1.4 Hardware Considerations

This document does not address the hardware configuration of the Controlatron tube as it is placed or operated within the test fixture. Please note that the operation of the Controlatron Test Suite Software within the facility is considered a hazardous operation due to high voltages used and the production of radioactive flux. Appropriate safety procedures must be in place and adhered to during the operation a Controlatron neutron tube.

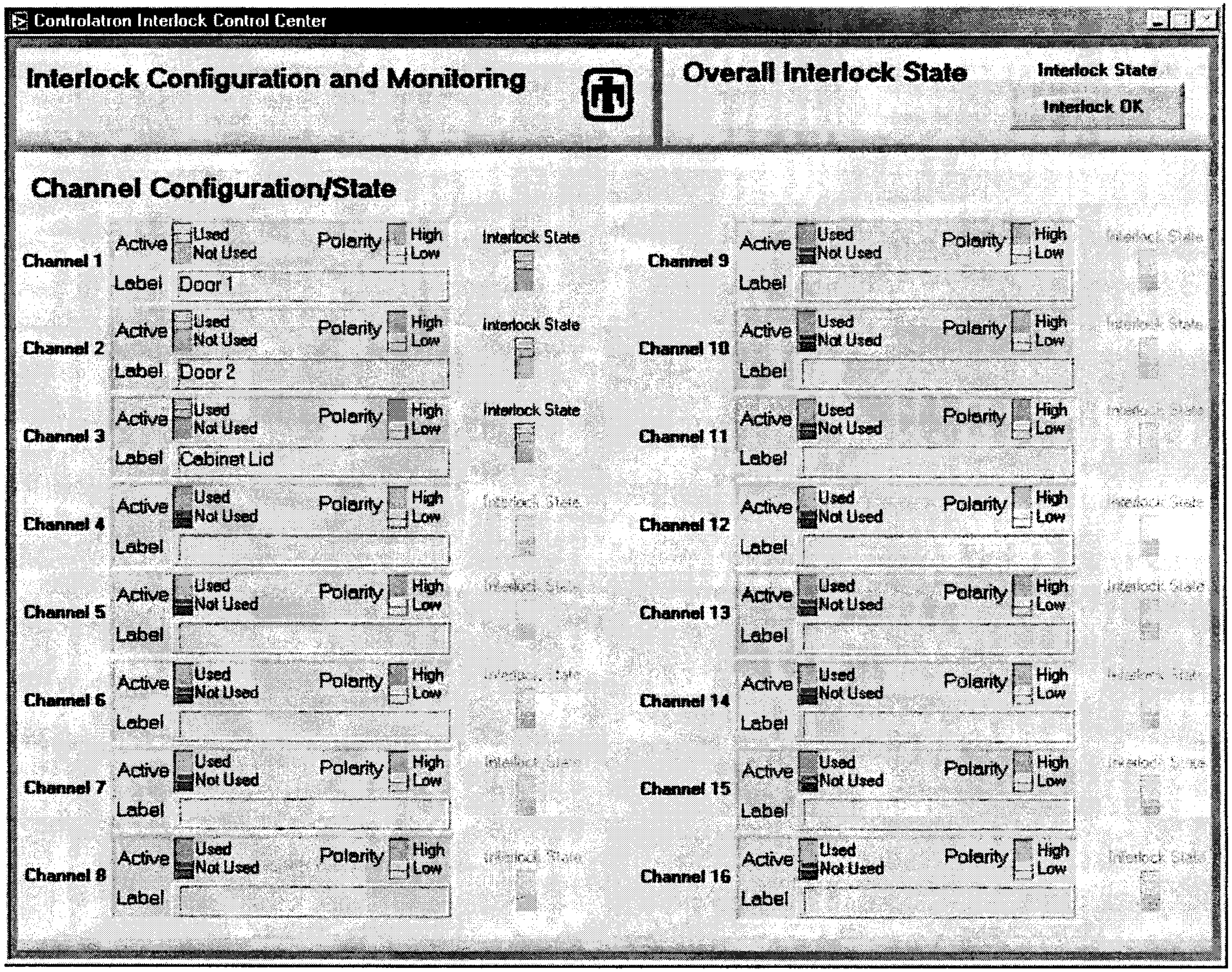
1.5 Configuring the Interlock System

The interlock system is configured on the Controlatron Interlock Control Center Screen. This screen allows the user to alter the configuration of the interlocks, view the status of each interlock and the overall interlock state.

The Interlock system uses the first 16 lines of the digital I\O card. The user can select which channels to use with the Active button on the channel. Name the interlock channel and set the polarity of a passing interlock (whether high or low is a passing interlock). The Interlock State indicator shows if the interlock is OK or failed.

The overall interlock state is passing when all of the active interlocks are passing (green).

- The example above shows three active interlocks (door1, door2, and cabinet lid), which are passing when the line is less than 3 volts.
- The Digital I\O card will sense logic high at any voltage above 3V and can accept any voltage up to 28V.



Controlatron Interlock Control Center

Interlock Configuration and Monitoring

Overall Interlock State

Interlock State

Interlock OK

Channel Configuration/State

Channel 1	Active: Used / Not Used	Polarity: High / Low	Label: Door 1	Interlock State
Channel 2	Active: Used / Not Used	Polarity: High / Low	Label: Door 2	Interlock State
Channel 3	Active: Used / Not Used	Polarity: High / Low	Label: Cabinet Lid	Interlock State
Channel 4	Active: Used / Not Used	Polarity: High / Low	Label:	Interlock State
Channel 5	Active: Used / Not Used	Polarity: High / Low	Label:	Interlock State
Channel 6	Active: Used / Not Used	Polarity: High / Low	Label:	Interlock State
Channel 7	Active: Used / Not Used	Polarity: High / Low	Label:	Interlock State
Channel 8	Active: Used / Not Used	Polarity: High / Low	Label:	Interlock State
Channel 9	Active: Used / Not Used	Polarity: High / Low	Label:	Interlock State
Channel 10	Active: Used / Not Used	Polarity: High / Low	Label:	Interlock State
Channel 11	Active: Used / Not Used	Polarity: High / Low	Label:	Interlock State
Channel 12	Active: Used / Not Used	Polarity: High / Low	Label:	Interlock State
Channel 13	Active: Used / Not Used	Polarity: High / Low	Label:	Interlock State
Channel 14	Active: Used / Not Used	Polarity: High / Low	Label:	Interlock State
Channel 15	Active: Used / Not Used	Polarity: High / Low	Label:	Interlock State
Channel 16	Active: Used / Not Used	Polarity: High / Low	Label:	Interlock State

Figure 3. (Previous Page) Controlatron Interlock Control Center Panel

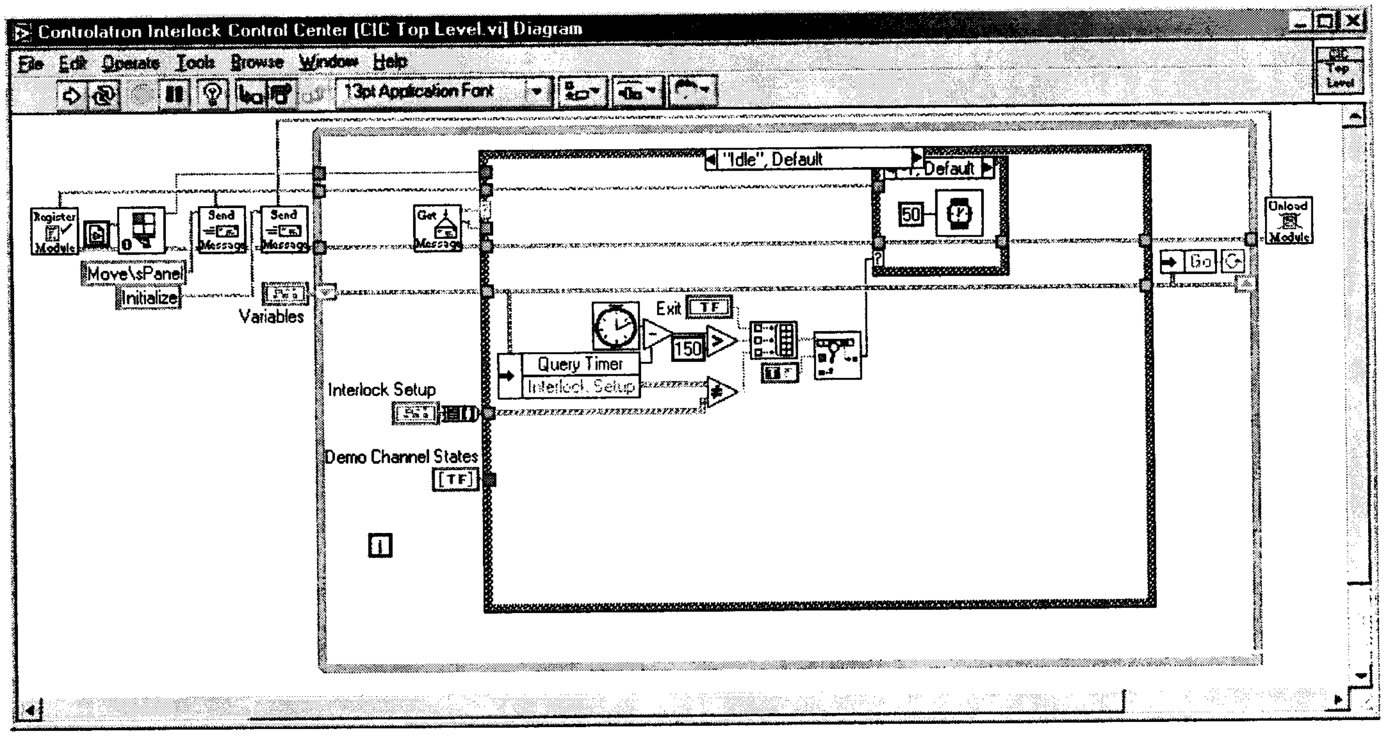


Figure 4. Controlatron Interlock Control Module Idle Case

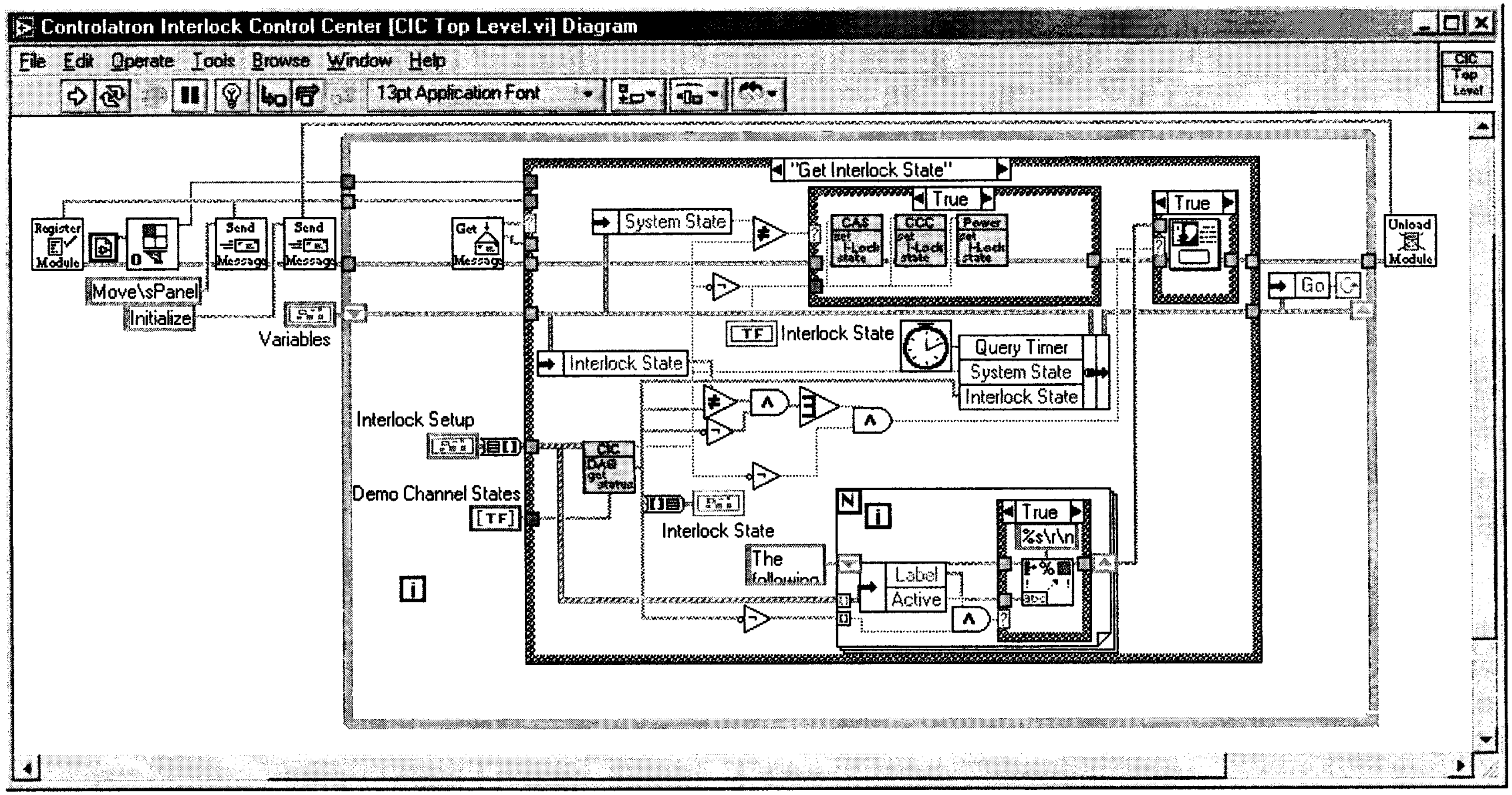


Figure 5. Controlatron Interlock Control Module 'Get Interlock State' Case

- When changes are made, they are automatically saved and the interlock setup will be the same the next time the system is used.

1.5.1.1 Overview

This module monitors the interlocks and informs the other modules of the interlock state so shut down will occur in the event of an interlock fault.

1.5.1.2 Operation

The Idle case determines if user changes of the configuration have occurred and if so, it replaces the interlock setup variable data.

The CIC DAQ get interlock status VI, calls the DIO card and checks the first 16 input channels of the DIO card. If the Interlock states change, it sends the message "set interlock state" to all other modules telling of its condition.

1.5.1.3 Hardware and Instruments

- **DIO 6527**

  The Interlock Module utilizes the National Instruments DIO 6527 card. This card is an optically isolated DIO card that can accept 24 channels of 0–28 V Input (above 2V is a logic high) and can switch 24 channels of digital outputs 0–60 V up to 0.120 amps.

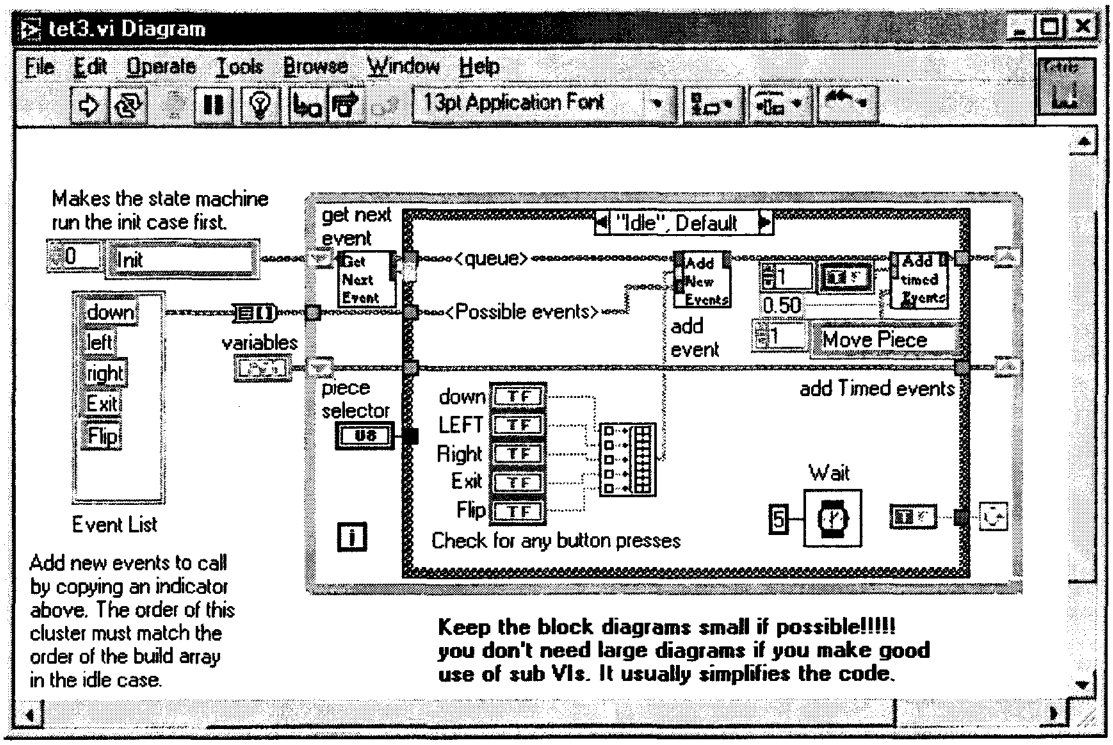
The Interlock Module only calls the DIO card drivers. These are basic DAQ Read Digital Port calls standard to LabVIEW that control any series of DIO card (so long as the hardware has enough lines).

1.6 Message Manager Architecture Description and Tutorial

PrimeCore Systems utilizes one of two basic architectures for implementing GUIs and complex sub-programs: the queued state machine, and the message manager system. They are both similar in concept and very flexible. The major difference is that the queued state is primarily designed to be a single stand-alone program and the message manager is designed for running multiple programs that can all communicate with each other through a standardized interface. The Queued state is generally better for novice programmers and simple systems, while the message manager is targeted toward large complex systems and distributed systems. This document is intended as a reference guide to learn to troubleshoot and implement the Message Manager based programs in general terms. Open the VI "Sample User Interface.vi" which is used as the example for this tutorial.

1.6.1 What is a state machine?

In its simplest terms a state machine is just a case structure inside a while loop. It is a handy way to piece together an application. There are often a large number of cases, each responsible for taking a particular action. Additionally, there is always at least one case that runs when nothing is happening and checks to see if the program should do something. The basic theory is that when nothing is pressed or nothing is going on, all the program does is check to see if it should be doing something. If it needs to do something, the program figures out what it should do and calls up the case or cases responsible for doing that task and those cases then perform whatever needs to be done. In the example below, when the user presses the left button for instance, the program goes to the left case and moves the piece to the left if it can. This creates a simple yet very flexible architecture for programming virtually any user interface in LabVIEW.



Makes the state machine run the init case first.

Add new events to call by copying an indicator above. The order of this cluster must match the order of the build array in the idle case.

Keep the block diagrams small if possible!!!!! you don't need large diagrams if you make good use of sub VIs. It usually simplifies the code.

Figure 6. Example Queued State Machine 'Idle State'

1.6.2 What is a 'State?'

Each Case in the case structure encased in the while loop is considered a 'State.' Each State can be called to perform an action based upon a user request, managed in the 'Idle' State, or programmatically by any other available state. Each state has a unique name and should perform one logical group of functions. An example of a reasonable state is a 'Write Data File' state. This example state would manage all of the necessary functions to create a file, write data to the new file, and then close the systems resources used in the file creation. In the application, this one state could manage the entire data file writing requirements. An advantage to the developer is the inherent modularity in the architecture.

1.6.3 What is the Message Manager?

There are many ways to implement a state machine. The message manager serves as the message queue and communications portal for a state machine or a collection of state machines running in concert. There is a set of utility VIs called the message manager that handles all of this communication. On top of these utility VIs, the message manager is a philosophy and method of programming architecture that is centered around multiple state machine VIs running concurrently, passing data and actions to each other through the message manager utilities. Each separate state machine VI is also called a "module" since if you design the code well, then each major function or sub-system will have its own "module" that can be reused and debugged by itself, independent of other modules.

1.6.3.1 The Key Elements to the Message Manager are:

- *Modules:* These are state machine programs that serve as user interfaces, drivers, or programs to control a system. These are developed as state machines by the programmer that utilize the message manager. These are the basically all the high level components of a LabVIEW application created using the message manager architecture.

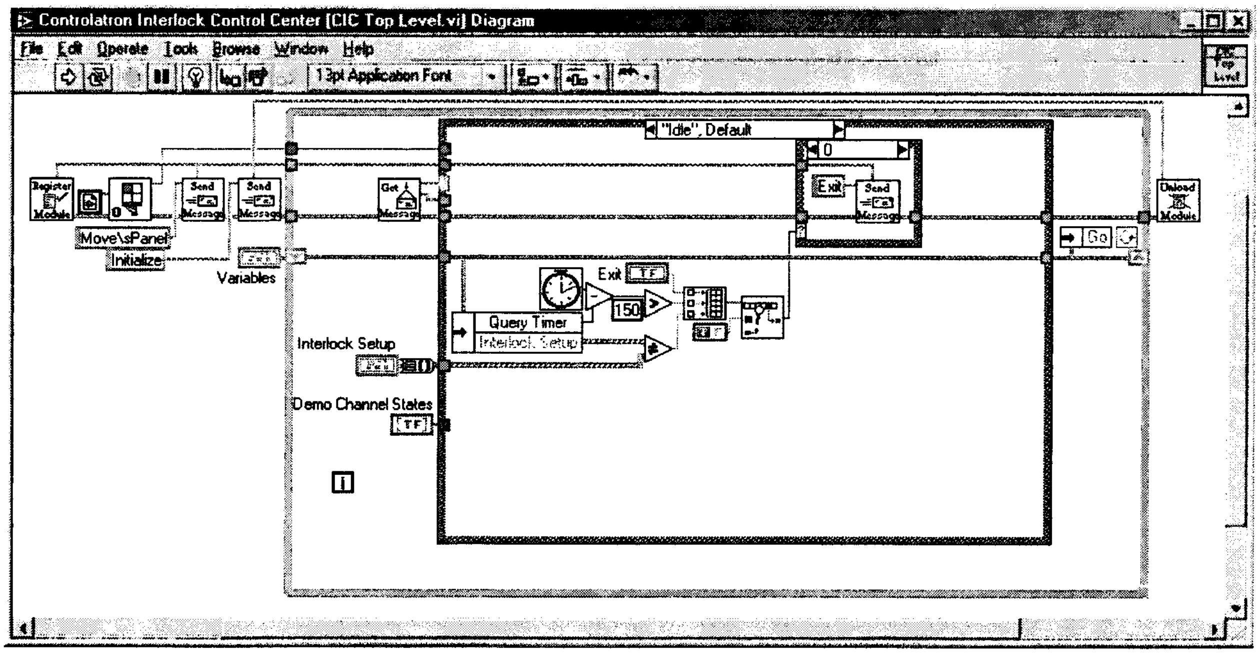


Figure 7. CIC Top Level Diagram (Message Manager Idle Event)

Figure 7 is the Idle Event of the CIC Top Level.vi found in the Controlatron Application. Notice the differences between this diagram and the Queued State Machine Diagram illustrated in Figure 6. There are events prior to executing the main 'While Loop' that are required to register this VI as a Module.

- *Module Manager:* This is a behind the scenes program that is the heart of the message manager. It contains queues for modules, initializes, or destroys queues.

- ***Register Module:*** This VI serves to initialize the message manager and to initialize the module that this VI is placed in. Normally, this is the first VI called in a module. This tells the module manager to allocate a queue for this module.

- ***Get Message.VI:*** Every module should use the Get Message.VI to run the state machine. It checks the queue for messages for that module and feeds a set of strings (the message) to the case structure of the state machine. It basically runs every state machine.

- ***Send Message.VI:*** This VI is used to send a message to a module that is retrieved later by the Get Message.VI. To use it, specify the module you want to send the message to (by name), the event (the case you want to run), and the event buffer (any data you want to send with the message).

1.6.3.2 What is a Message?

A message is a set of data that tells a module what state to go to and provides data for that state to run if necessary. In simple terms, the message Manager VIs use 3 strings to make up a message: the event, the event buffer, and the source. This elaborates on most state machines in that a string defines the state, and it also contains data to run with the state. This often proves to be more useful and flexible than just using a queue of states and transferring data and variables separately.

- ***Event:*** This is a string that tells a module what case (event) to run. It is output by the "get message.vi" and should be wired in to the case selector of the module. The event should match the name of a case in the module

- ***Event Buffer:*** This is a string that can contain data to be used in conjunction with the event. You can represent any type of data as a string, that is why that was selected. This allows you to transfer data between different modules. This is optional, you don't always need data, sometimes it is enough just to tell the module what event to go to.

- ***Source:*** This tells what module the message came from. This is useful for troubleshooting when you have multiple modules communicating with each other.

1.6.3.3 How do I add events to the message queue?

Since all top-level programs running in a message manager based system use the "get message.vi" to run the state machine, you must use the message manager VIs to control the actions of your program.

To send a message to run a case from inside the same module you can use the "send message.vi" or the "deliver message.vi". The key difference between these two is that the send messages puts the message at the end of the stack, and deliver message bumps the message to the top of the stack (forces it to be the next event run). If you are sending a message to the module you are in, then you don't need to wire in the destination module input.

1.6.3.4 Packaging and Un-packaging Data for a Message

Using the message manager, one of the most time consuming and difficult tasks is to convert the data to string for the event buffer data. Using the following strategies can help packaging the data and remain flexible and change tolerant.

For Simple data such as a Boolean, array, or numeric it is easiest to use the type cast function to pack data to a string for sending through the message manager.

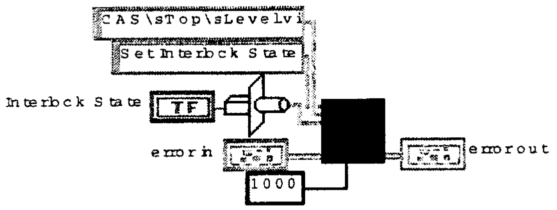


Figure 8. Packing Data For Message Management

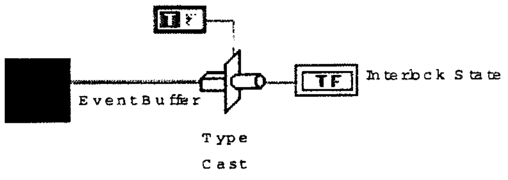


Figure 9. Unpacking Data Received in a Message

For sending more complex data structures, it is usually easiest to package the data using the Flatten and Un-flatten from string functions. These will convert complex clusters or other data types to a string. The best way to do this is to use a cluster saved as a type-def control. Use the type-def control as the key to flatten and un-flatten the data as shown below. Using a type-def means that if you change the type-def data type, it will automatically update all the packaging and un-packaging functions you create for you and make the code more robust and tolerant to change.

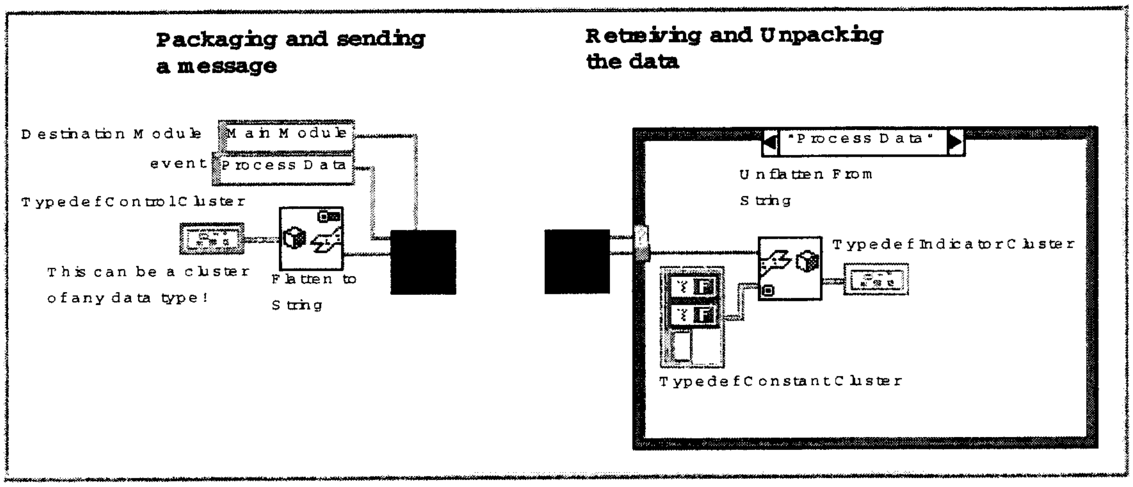


Figure 10. Packing and Unpacking Data Summary

1.7 Overall Architecture of a Message Manager Based Program

The basic concept is to break up a large application into several modules that handle a particular task or sub-systems. That way you may be able to reuse modules and also have different programmers working on different modules so you can break up a project into smaller pieces easier. All the modules communicate to each other through the message manager.

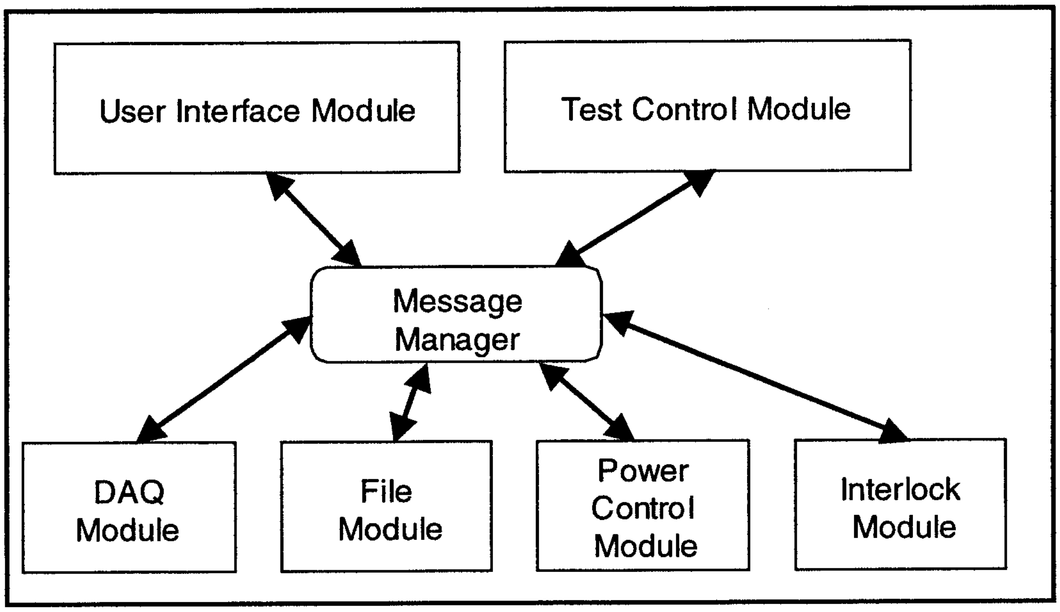


Figure 11. General Message Manager Architecture

Through the message manager, any module can communicate with any other module, but in general it is usually a good idea to have one or two high level modules that generally control the application and call a series of modules that perform the low level tasks of the system. If the system isn't very complex, then it is simplest to use a single high-level application to control the logic of the control system, and many modules to perform the sub-tasks and control sub-systems. If it is a large application, you may wish to create a more complex hierarchy and object-orient the design of the code. See the discussion on object orientation at the end of the tutorial.

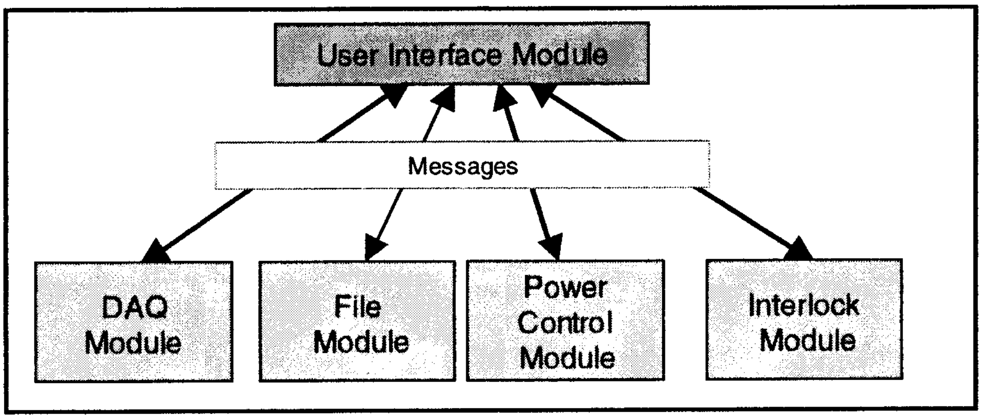


Figure 12. Module Responsibility Layout

1.8 Remote Message Manager

This basic architecture lends itself well to distributed applications running on different machines. The remote message manager just adds a layer to calls going to another computer. The remote message manager.vi is the engine that manages the remote connections. Once you establish a connection to another machine, then you can use the "send remote message.vi" instead of the send message.vi to communicate with a remote machine. Architecturally, it allows multiple programs running on different machines to communicate through a common interface. In future versions of the message manager it is planned to implement this more transparently without separate "remote" calls.

1.9 Advantages of a State Machine in General

- **Size:** It allows you to make a large program with a small block diagram.
- **Efficiency:** The state machine tends to efficient if implemented properly.
- **Troubleshooting:** Since every action takes place pretty much in its own event (or case) it tends to be far easier to figure out what is going on than other architecture styles. Since the diagrams are smaller and more segmented, the light bulb and single step features work better than with large diagrams. They also tend to produce less "race conditions" by nature since everything is divided into small cases.
- **Standardization:** If every developer in a group uses it, it is much easier to debug and augment someone else's code.
- **Flexibility:** You can create any sequence of actions you desire with a state machine.

1.10 Advantages of the Message Manager

- **Standardization:** The best feature is that it is a flexible architecture that can be adapted to just about any program. It is a more intricate philosophy than a queued state machine.
- **Parallel Operation:** This allows you to run many sub systems together, all running in parallel with their own loop update rates.
- **Scalability and Reusability:** If you are careful in your top-level design, you can create modules that can be used in other programs with a minimum of rewriting code. Also, if you design your modules well, you can have stand-alone sub programs for sub-systems independent of your top level application, or you can use access functions to very quickly build new programs.
- **Common Data Portal:** All data that passes between different sub-programs uses the same interface. This reduces complexity and makes the code easier to understand.

1.11 Disadvantages of the Message Manager

- ***Speed of Object Oriented Code:*** Object oriented code requires more overhead in accessing data and the functionality of the system. It generally makes better code, but there are more layers to access data and function, thereby slowing down the code. The message manager is pretty well optimized, but if you are transferring large amounts of data, it can slow the system down.

- ***More difficult to troubleshoot than a simple state machine:*** It is harder to figure out what is going on with multiple modules running together than a single state machine. Design how the modules interact before integrating the modules together! Also, be sure to document how the system modules interact, otherwise it is very difficult to figure out what is going on a year down the road.

- ***More Up-Front coding time:*** It takes longer to write and get it working, especially if you use access functions and object-orient it. However, if you design it well, it should be easier down the road to troubleshoot and add functionality. Also, it should speed up projects with multiple developers since it is by nature modular. It is not recommended for a simple DAQ application, but it is recommended for complex systems involving multiple instruments and\or functionality.

1.12 Object Orientation Using the Message Manager

The message manager architecture is well suited to making LabVIEW code more Object Oriented. You can't truly make LabVIEW code completely object oriented, but using the strategies below; you can accomplish most of the key properties of object-oriented code.

With the message manager architecture, the key to accomplishing object orientation is to first create modules for each major sub-system. The modules will be the "objects". The module should contain all of the functionality for the sub-system or user interface. If you are really gung ho about object orientation, you can even adapt the modules to create copies of themselves and refer to them by reference for using multiple objects that are the same. That is beyond the scope of this discussion, but certainly possible.

Additionally, all of the data involved with the sub-system should reside in the module (no globals or shortcuts) and should only be accessed through function calls to the module. If you need the data from a module in the user interface or another module, you should create an access function that reads it from the module itself and displays it through the user interface (or wherever). That way your data is protected and can only be accessed through proper channels. Also, write access functions for all commands, functions, setups, or whatever you need to command the module to do. Like the data encapsulation, it forces the programmer to go through proper channels to command the module to perform an action.

The sample uses both simple methods and some of the object-style access functions to call the modules so you can see the benefits of writing access functions for your modules.

So an object should consist of a set of VIs and type-def controls as follows.

- ***The Module VI:*** This is the message manager based state machine that contains all of the functionality code of the sub-system. It serves as the "object." It also contains all the data associated with that module, preferably in a shift register.

- ***Access Functions:*** These functions call the module through the message manager and perform a function or to return data. These functions encapsulate the module; so write access functions for ***everything*** you need to do with the module object. These are how to utilize a module if you want it object oriented. In other words, don't use the send message or package or un-package the data for another module in the diagram of a module, use access functions for everything.

  Figure 13 is a block diagram for a simple access function. It packages the data, tells it what case and module to go to so all the programmer needs to do is to drop this access function down and wire in the Arm Boolean. This helps encapsulate the functionality of the module.

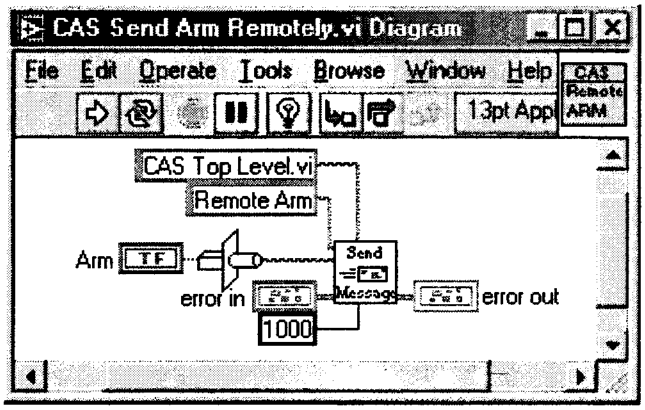


Figure 13. Example 'Access Function'

- ***Type-Def Controls:*** Use type definitions or strict type-def controls for all complex data structures. That way your modules and access functions change together automatically if you have to change your data types. Use clusters and use unbundled and bundle by name wherever possible so that the code does not break when you add a variable. Try not to rename or delete variables unless necessary.

1.13 Queued State Machine Architecture Description and Tutorial

PrimeCore Systems utilizes one of two basic architectures for implementing GUIs and complex sub-programs: the queued state machine, and the message manager system. The more simple architecture used is the Queued State Machine.

1.13.1 First of all, what is a state machine?

In its simplest terms a state machine is just a case structure inside a while loop. There are often a pretty large number of cases, each responsible for taking a particular action. Additionally, there is always at least one case that runs when nothing is happening and checks to see if the program should do something. So the basic theory is that when nothing is pressed or nothing is going on, all the program does is checks to see if it should be doing something. If it needs to do something, the program figures out what it should do and calls up the case or cases responsible for doing that task and those cases then perform whatever needs to be done. In the example below, when the user presses the left button for instance, the program goes to the left case and moves the piece to the left if it can. This creates a simple yet very flexible architecture for programming virtually any user interface in LabVIEW.

1.13.2 What is a queued state machine?

There are many ways to implement a state machine; the queued state method utilizes a queue that is an array of strings to control the state machine. When an action is evoked, the state machine adds a string element to the queue array. The next time the loop cycles the string element is fed into the case structure using the Get Next Event VI. Using an array of strings allows for a whole sequence of events to be loaded up in the queue and run one after another for cascaded events.

1.13.3 How do I add events to the queue?

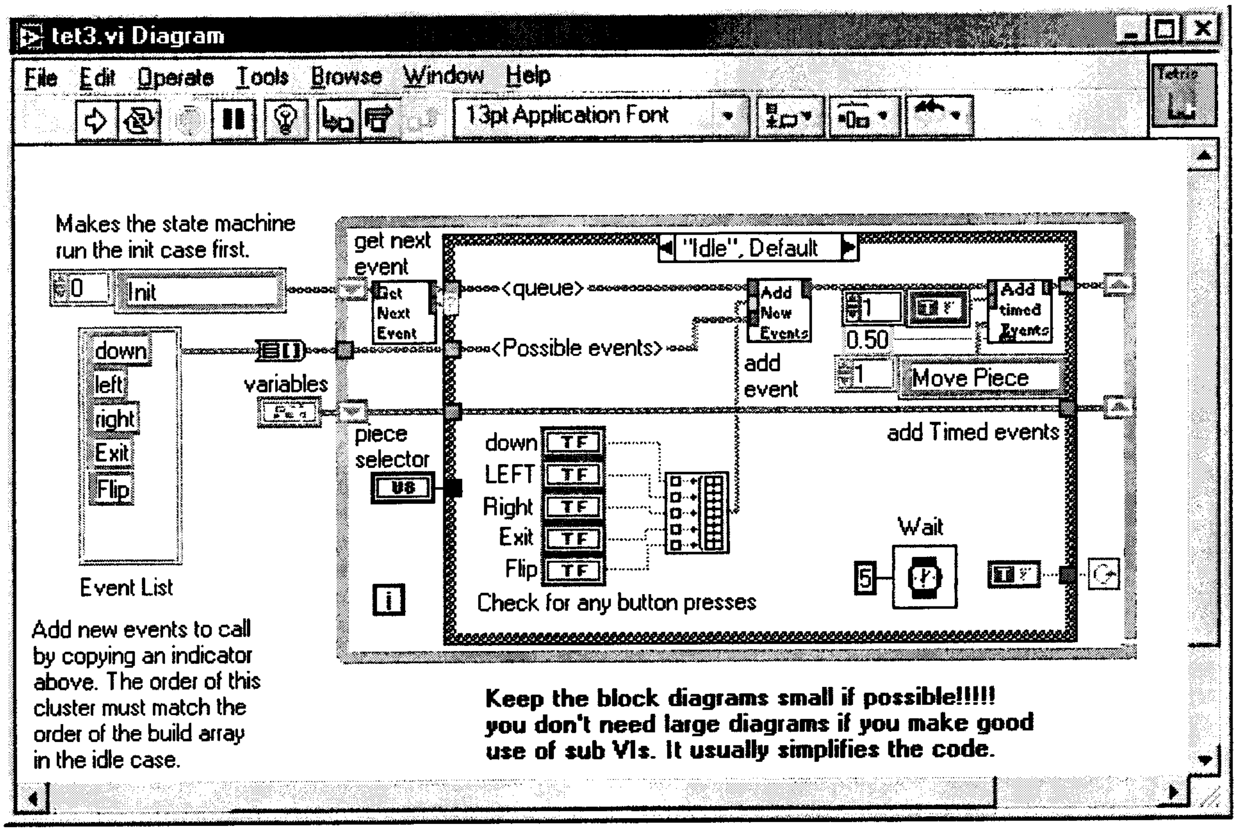
Add New Events and Add Periodic Events VIs can be used to add events to the queue. The input elements are a list of events (as an array of strings) that are potentially available and a Boolean array that tells the sub-VI weather each event should run or not (a true causes that event to be added to the queue). The array of strings just needs to match up element for element with the array of Booleans.

Other ways of taking action can include comparing a value of an indicator to its previous value. If they are not equal, the Boolean is true and an action should occur.

Figure 14 shows an example of using a cluster and a shift register to contain variables. In the idle case it checks the value of the piece variable and if it changes it will feed a true to the add new events case and cause an action to happen. That way you can use the state machine to control events based on changing values or Booleans.

Also, if you want to call one case right after another you can use the build array function to call another event. That is useful for cascaded actions or for sequential operations.

The Build array in the upper right side causes the update piece case to be called right after the move piece case is called.



Add new events to call by copying an indicator above. The order of this cluster must match the order of the build array in the idle case.

Keep the block diagrams small if possible!!!!! you don't need large diagrams if you make good use of sub VIs. It usually simplifies the code.

Figure 14. Adding Events to the 'Queue' of a Queued State Machine

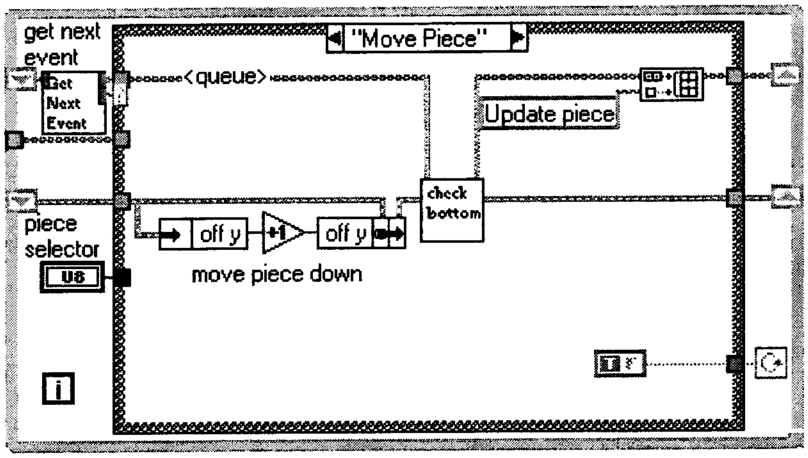


Figure 15. Programmatically adding events to the Queue

1.13.4 Timed Events

Using the Add Periodic Events VI, you can cause an action to occur every time interval. In this example the move piece case gets called every half a second to drop the piece. It works just like the Add New Events except that it only adds events when the timer goes off.

1.13.5 What can I do in a "State?"

Everything that happens should pretty much have its own case or state. You can update the display, send a command to an instrument, or read in information in a case.

1.13.6 Advantages of a State Machine

- **Size:** It allows you to make a large program with a small block diagram.
- **Efficiency:** The state machine tends to be efficient if implemented properly.
- **Troubleshooting:** Since every action takes place pretty much in its own event (or case) it tends to be far easier to figure out what is going on than other architecture styles. Since the diagrams are smaller and more segmented, the light bulb and single step features work better than with large diagrams.
- **Standardization:** If every developer in a group uses it, it is much easier to debug and augment someone else's code.
- **Flexibility:** You can create any sequence of actions you desire with a state machine.

(INTENTIONALLY LEFT BLANK)

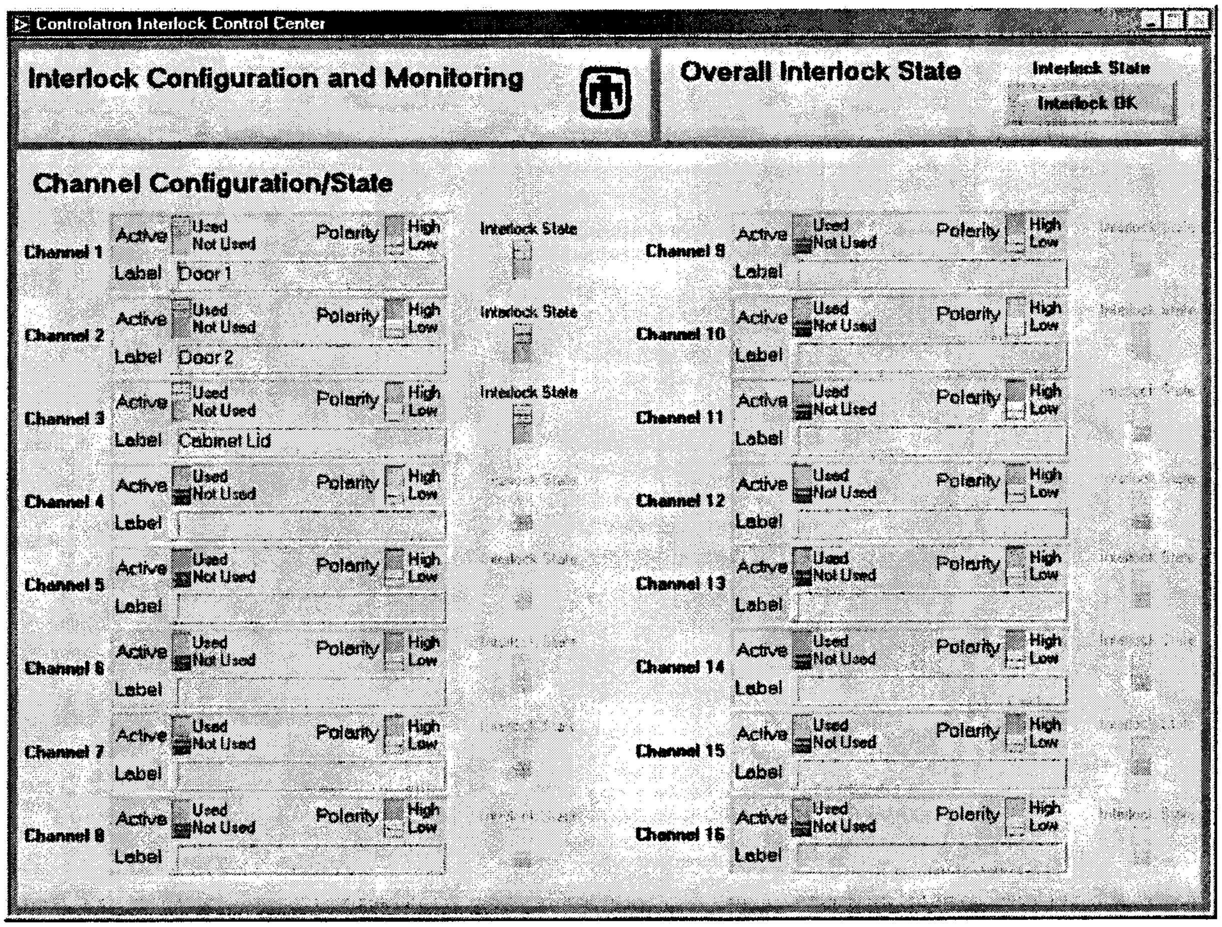
2. Controlatron Interlock Control System

2.1 CIC Top Level.vi

This Vi is a module in charge of checking the interlocks of the Controlatron test facility. It reports the interlock status directly to all the other modules.

2.1.1 Connector Pane

2.1.2 Front Panel



Controlatron Interlock Control Center

Interlock Configuration and Monitoring

Overall Interlock State — Interlock State: Interlock OK

Channel Configuration/State

Channel	Active	Polarity	Label	Interlock State
Channel 1	Used / Not Used	High / Low	Door 1	Interlock State
Channel 2	Used / Not Used	High / Low	Door 2	Interlock State
Channel 3	Used / Not Used	High / Low	Cabinet Lid	Interlock State
Channel 4	Used / Not Used	High / Low		
Channel 5	Used / Not Used	High / Low		
Channel 6	Used / Not Used	High / Low		
Channel 7	Used / Not Used	High / Low		
Channel 8	Used / Not Used	High / Low		
Channel 9	Used / Not Used	High / Low		
Channel 10	Used / Not Used	High / Low		
Channel 11	Used / Not Used	High / Low		
Channel 12	Used / Not Used	High / Low		
Channel 13	Used / Not Used	High / Low		
Channel 14	Used / Not Used	High / Low		
Channel 15	Used / Not Used	High / Low		
Channel 16	Used / Not Used	High / Low		

2.1.3 Controls and Indicators

Interlock Setup Interlock State Indicators show the current state of the interlock channels adjacent to them.

Channel 1 Determines if this channel is actively monitored as part of the Interlock System.

Active Determines if this channel is actively monitored as part of the Interlock System.

Polarity Determines the polarity state of the channel assigned to this interlock.

To successfully achieve an interlock, the channel must be active and the signal read by the DAQ Card must match the polarity assigned by this control.

Label This is the arbitrary name of the interlock given by the user to define this channel.

Channel 2 Determines if this channel is actively monitored as part of the Interlock System.

Active Determines if this channel is actively monitored as part of the Interlock System.

Polarity Determines the polarity state of the channel assigned to this interlock.

To successfully achieve an interlock, the channel must be active and the signal read by the DAQ Card must match the polarity assigned by this control.

Label This is the arbitrary name of the interlock given by the user to define this channel.

Channel 3 Determines if this channel is actively monitored as part of the Interlock System.

Active Determines if this channel is actively monitored as part of the Interlock System.

Polarity Determines the polarity state of the channel assigned to this interlock.

To successfully achieve an interlock, the channel must be active and the signal read by the DAQ Card must match the polarity assigned by this control.

Label This is the arbitrary name of the interlock given by the user to define this channel.

Channel 4 Determines if this channel is actively monitored as part of the Interlock System.

Active Determines if this channel is actively monitored as part of the Interlock System.

Polarity Determines the polarity state of the channel assigned to this interlock.

To successfully achieve an interlock, the channel must be active and the signal read by the DAQ Card must match the polarity assigned by this control.

Label This is the arbitrary name of the interlock given by the user to define this channel.

Channel 5 Determines if this channel is actively monitored as part of the Interlock System.

Active Determines if this channel is actively monitored as part of the Interlock System.

Polarity Determines the polarity state of the channel assigned to this interlock.

To successfully achieve an interlock, the channel must be active and the signal read by the DAQ Card must match the polarity assigned by this control.

Label This is the arbitrary name of the interlock given by the user to define this channel.

Channel 6 Determines if this channel is actively monitored as part of the Interlock System.

Active Determines if this channel is actively monitored as part of the Interlock System.

Polarity Determines the polarity state of the channel assigned to this interlock.

To successfully achieve an interlock, the channel must be active and the signal read by the DAQ Card must match the polarity assigned by this control.

Label This is the arbitrary name of the interlock given by the user to define this channel.

Channel 7 Determines if this channel is actively monitored as part of the Interlock System.

Active Determines if this channel is actively monitored as part of the Interlock System.

Polarity Determines the polarity state of the channel assigned to this interlock.

To successfully achieve an interlock, the channel must be active and the signal read by the DAQ Card must match the polarity assigned by this control.

Label This is the arbitrary name of the interlock given by the user to define this channel.

Channel 8 Determines if this channel is actively monitored as part of the Interlock System.

Active Determines if this channel is actively monitored as part of the Interlock System.

Polarity Determines the polarity state of the channel assigned to this interlock.

To successfully achieve an interlock, the channel must be active and the signal read by the DAQ Card must match the polarity assigned by this control.

Label This is the arbitrary name of the interlock given by the user to define this channel.

Channel 9 Determines if this channel is actively monitored as part of the Interlock System.

Active Determines if this channel is actively monitored as part of the Interlock System.

Polarity Determines the polarity state of the channel assigned to this interlock.

To successfully achieve an interlock, the channel must be active and the signal read by the DAQ Card must match the polarity assigned by this control.

Label This is the arbitrary name of the interlock given by the user to define this channel.

Channel 10 Determines if this channel is actively monitored as part of the Interlock System.

Active Determines if this channel is actively monitored as part of the Interlock System.

Polarity Determines the polarity state of the channel assigned to this interlock.

To successfully achieve an interlock, the channel must be active and the signal read by the DAQ Card must match the polarity assigned by this control.

Label This is the arbitrary name of the interlock given by the user to define this channel.

Channel 11 Determines if this channel is actively monitored as part of the Interlock System.

Active Determines if this channel is actively monitored as part of the Interlock System.

Polarity Determines the polarity state of the channel assigned to this interlock.

To successfully achieve an interlock, the channel must be active and the signal read by the DAQ Card must match the polarity assigned by this control.

Label This is the arbitrary name of the interlock given by the user to define this channel.

Channel 12 Determines if this channel is actively monitored as part of the Interlock System.

Active Determines if this channel is actively monitored as part of the Interlock System.

Polarity Determines the polarity state of the channel assigned to this interlock.

To successfully achieve an interlock, the channel must be active and the signal read by the DAQ Card must match the polarity assigned by this control.

Label This is the arbitrary name of the interlock given by the user to define this channel.

Channel 13 Determines if this channel is actively monitored as part of the Interlock System.

- **Active** Determines if this channel is actively monitored as part of the Interlock System.
- **Polarity** Determines the polarity state of the channel assigned to this interlock.

  To successfully achieve an interlock, the channel must be active and the signal read by the DAQ Card must match the polarity assigned by this control.
- **Label** This is the arbitrary name of the interlock given by the user to define this channel.

Channel 14 Determines if this channel is actively monitored as part of the Interlock System.

- **Active** Determines if this channel is actively monitored as part of the Interlock System.
- **Polarity** Determines the polarity state of the channel assigned to this interlock.

  To successfully achieve an interlock, the channel must be active and the signal read by the DAQ Card must match the polarity assigned by this control.
- **Label** This is the arbitrary name of the interlock given by the user to define this channel.

Channel 15 Determines if this channel is actively monitored as part of the Interlock System.

- **Active** Determines if this channel is actively monitored as part of the Interlock System.
- **Polarity** Determines the polarity state of the channel assigned to this interlock.

  To successfully achieve an interlock, the channel must be active and the signal read by the DAQ Card must match the polarity assigned by this control.
- **Label** This is the arbitrary name of the interlock given by the user to define this channel.

Channel 16 Determines if this channel is actively monitored as part of the Interlock System.

- **Active** Determines if this channel is actively monitored as part of the Interlock System.
- **Polarity** Determines the polarity state of the channel assigned to this interlock.

  To successfully achieve an interlock, the channel must be active and the signal read by the DAQ Card must match the polarity assigned by this control.
- **Label** This is the arbitrary name of the interlock given by the user to define this channel.

Exit

Demo Channel States Values are used for demo purposes only.

- **Boolean**

Variables

- **Go**
- **Interlock Setup**
  - **Channel 1**
    - **Active**
    - **Polarity**
    - **Label**
- **Interlock State**
  - **Interlock State 1**
- **Query Timer**
- **System State**
- **Remote Mode**

Interlock State

- **Interlock State 1** Indicates the current interlock state of the adjacent defined interlock channel.
- **Interlock State 2** Indicates the current interlock state of the adjacent defined interlock channel.
- **Interlock State 3** Indicates the current interlock state of the adjacent defined interlock channel.
- **Interlock State 4** Indicates the current interlock state of the adjacent defined interlock channel.
- **Interlock State 5** Indicates the current interlock state of the adjacent defined interlock channel.
- **Interlock State 6** Indicates the current interlock state of the adjacent defined interlock channel.
- **Interlock State 7** Indicates the current interlock state of the adjacent defined interlock channel.
- **Interlock State 8** Indicates the current interlock state of the adjacent defined interlock

channel.

TF **Interlock State 9** Indicates the current interlock state of the adjacent defined interlock channel.

TF **Interlock State 10** Indicates the current interlock state of the adjacent defined interlock channel.

TF **Interlock State 11** Indicates the current interlock state of the adjacent defined interlock channel.

TF **Interlock State 12** Indicates the current interlock state of the adjacent defined interlock channel.

TF **Interlock State 13** Indicates the current interlock state of the adjacent defined interlock channel.

TF **Interlock State 14** Indicates the current interlock state of the adjacent defined interlock channel.

TF **Interlock State 15** Indicates the current interlock state of the adjacent defined interlock channel.

TF **Interlock State 16** Indicates the current interlock state of the adjacent defined interlock channel.

TF **Interlock State** The Armed button indicates and controls the state of the acquisition system. Arming the system can take up to 30 seconds.

Once the system is armed, the Manual Trigger will become available if the user is not in a predefined test mode.

Front Panel

I16 **left**

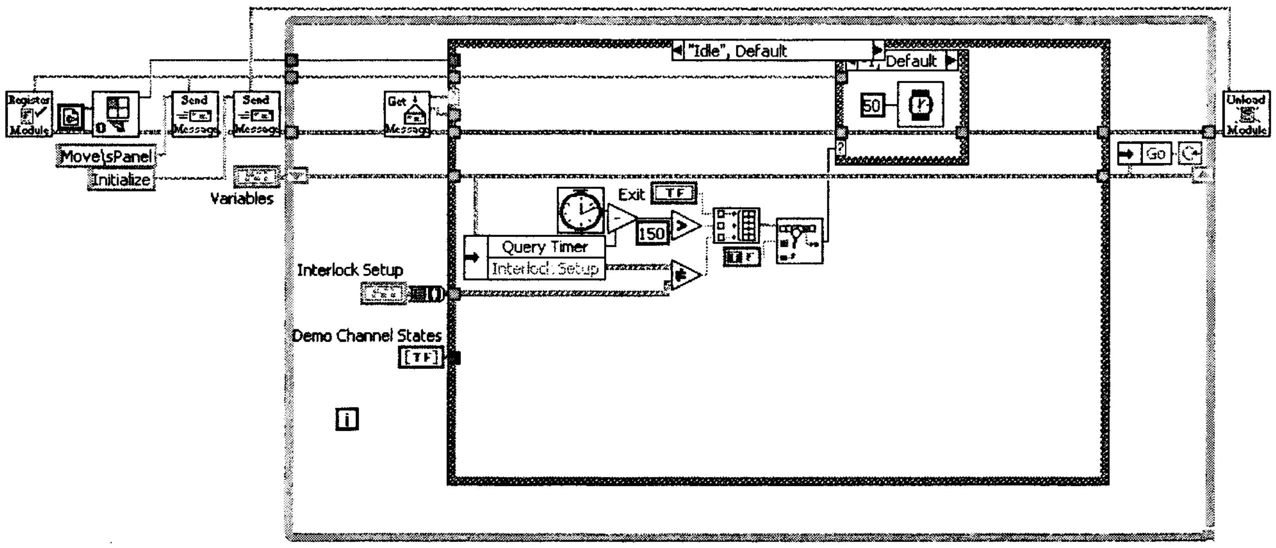
I16 **top**

I16 **right**

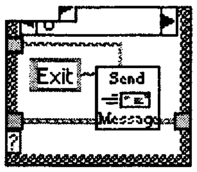
I16 **bottom**

2.1.4 Top Level Diagram

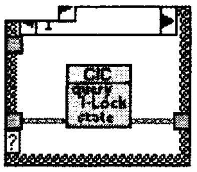
2.1.5 Controlatron Interlock Control



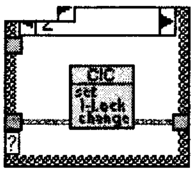
Case: 0



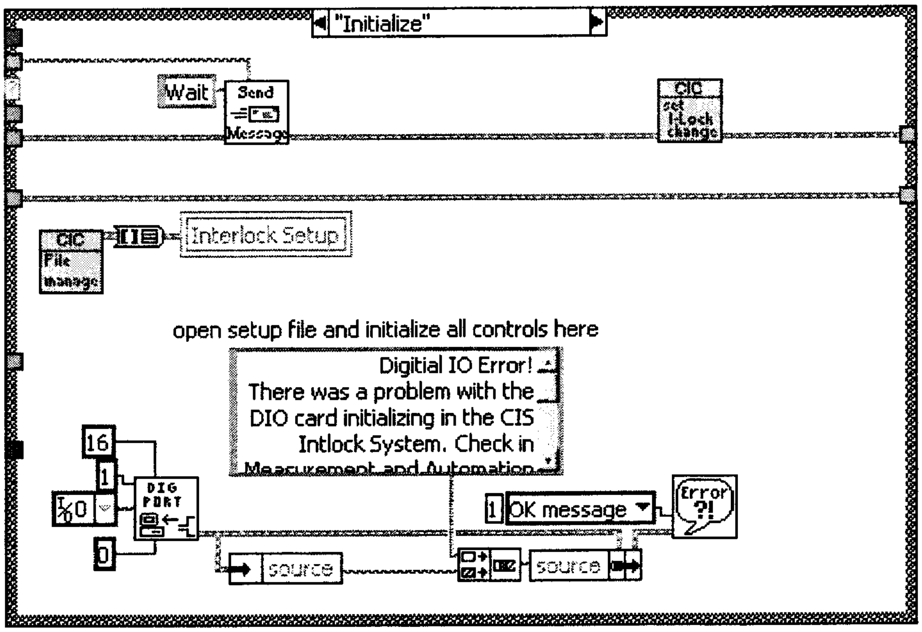
Case: 1



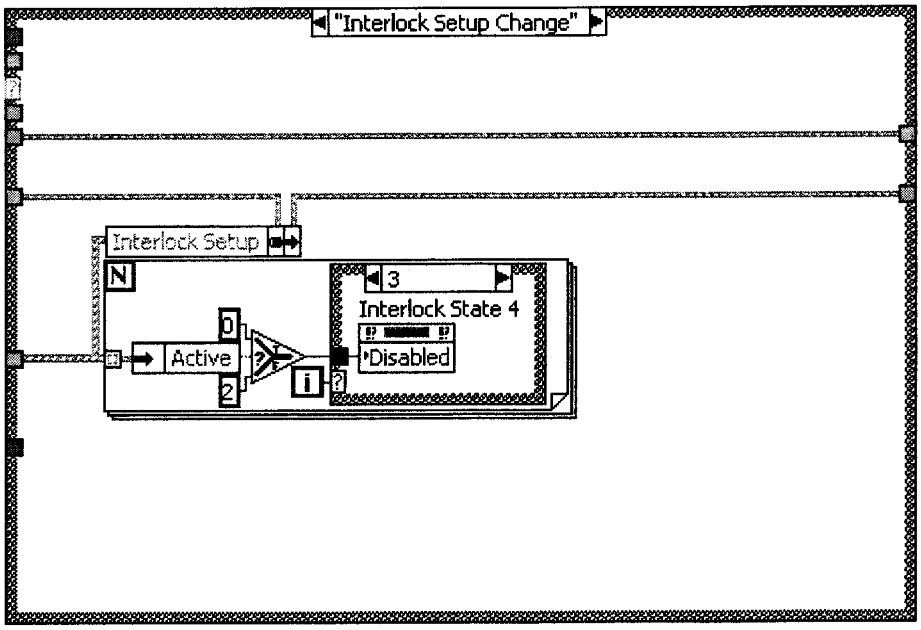
Case: 2



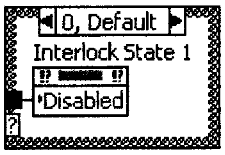
Case: Initialize


"Initialize"
Wait
Send Message
CIC set I-Lock change
CIC File manage
Interlock Setup
open setup file and initialize all controls here
Digitial IO Error! There was a problem with the DIO card initializing in the CIS Intlock System. Check in Measurement and Automation
16
1
0
DIG PORT
1
OK message
Error ?!
source
source

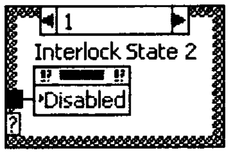
Case: Interlock setup Change



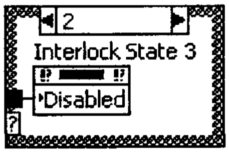
Case: 0,Default



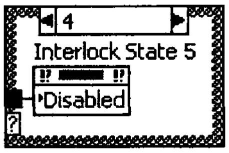
Case: 1



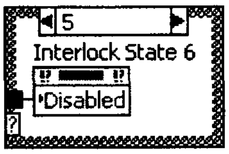
Case: 2



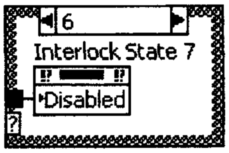
Case: 4



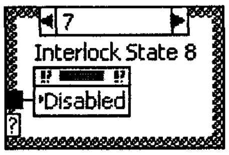
Case: 5



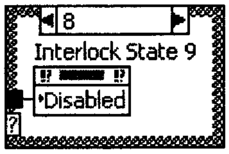
Case: 6



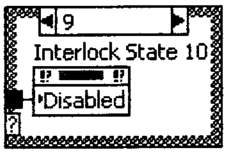
Case: 7



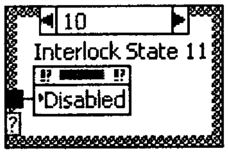
Case: 8



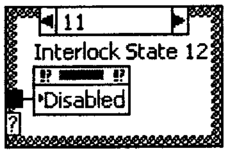
Case: 9



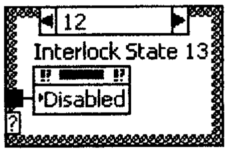
Case: 10



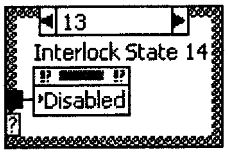
Case: 11



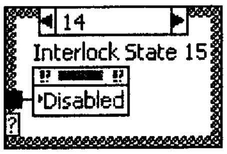
Case: 12



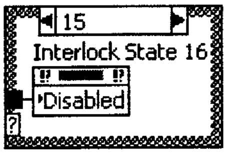
Case: 13



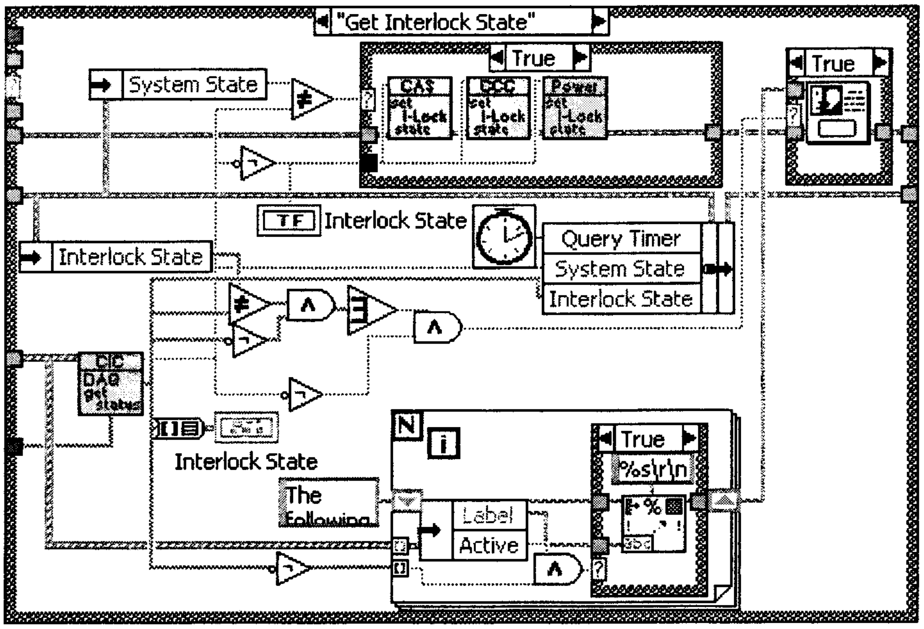
Case: 14



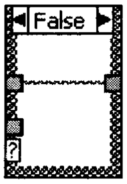
Case: 15



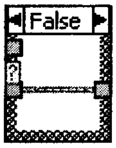
Case: Get Interlock State



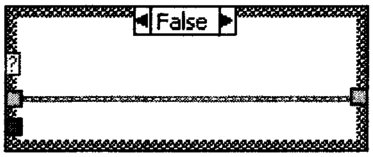
Case: False



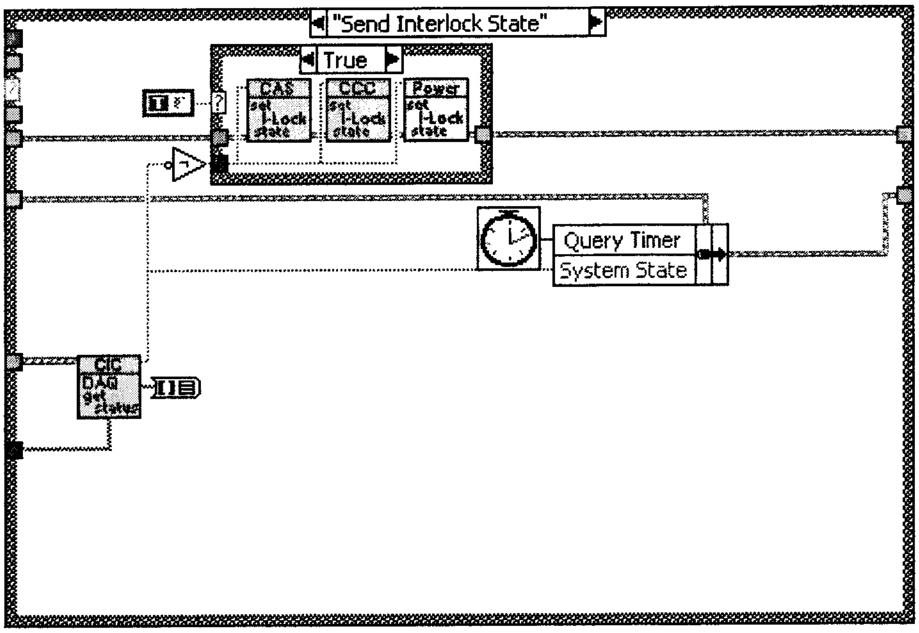
Case: False



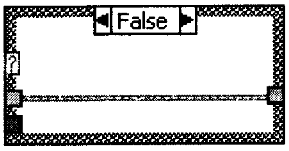
Case: False



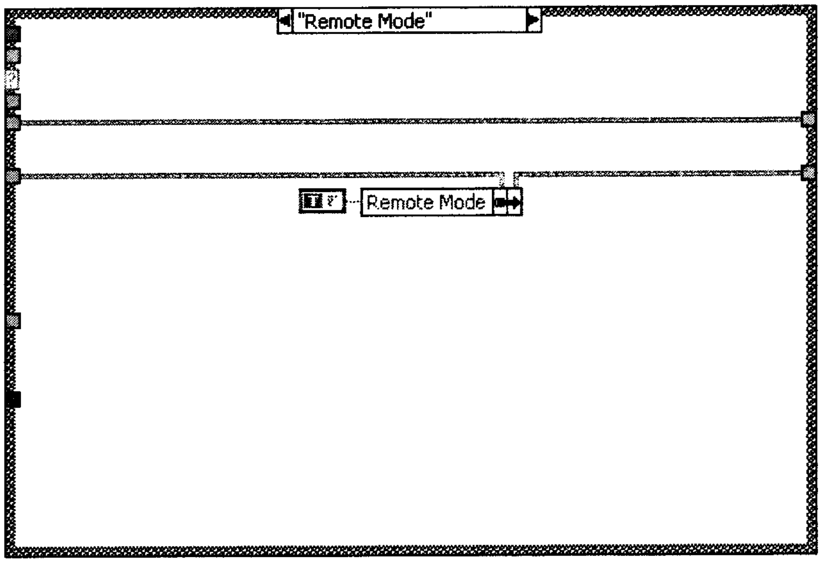
Case: Send Interlock State



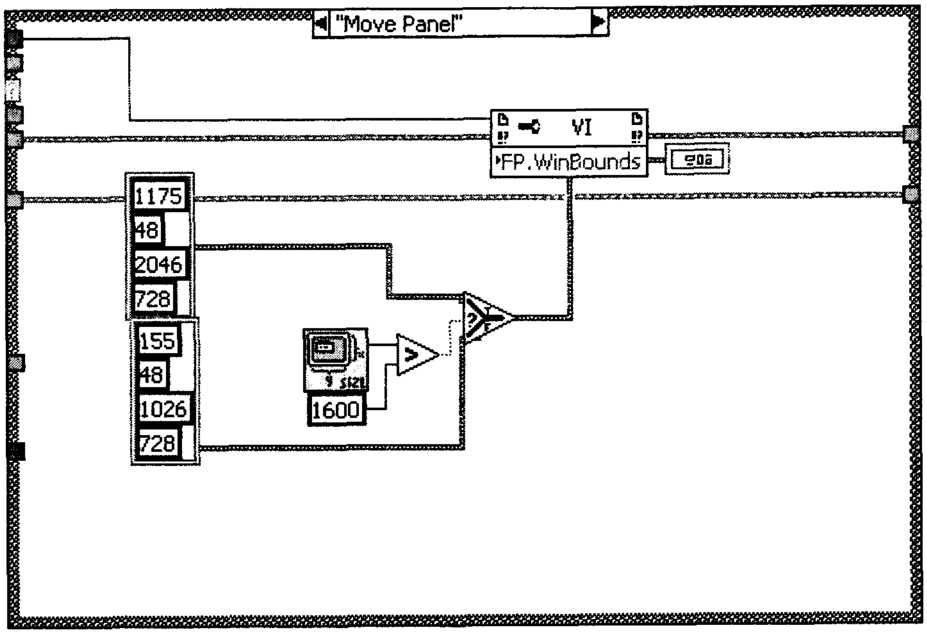
Case: False



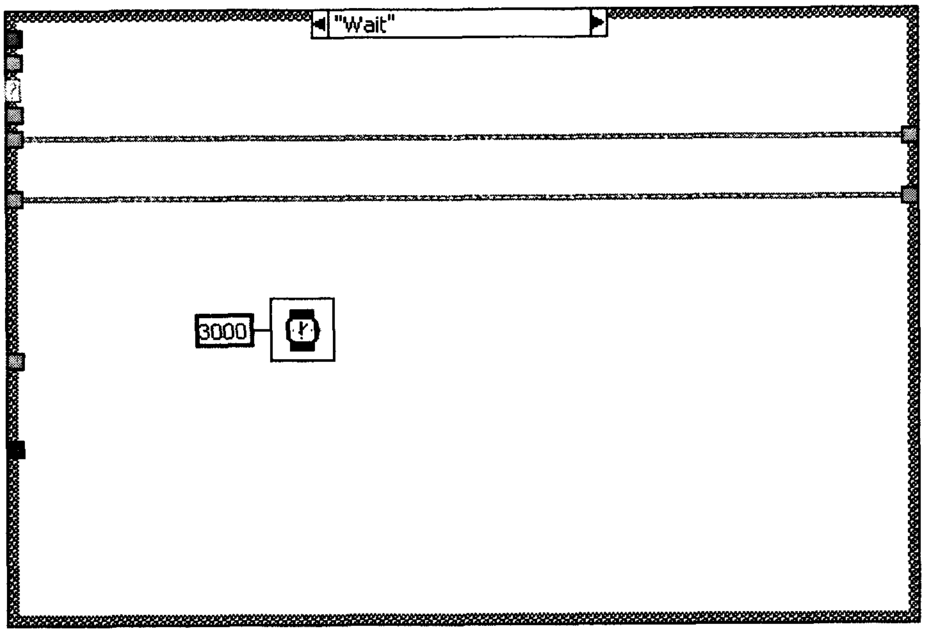
Case: Remote Mode



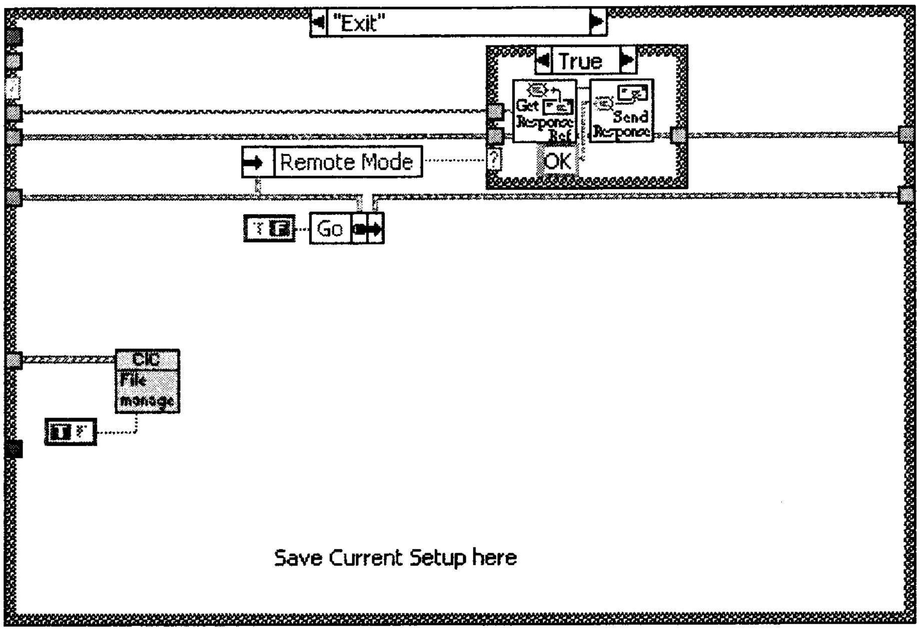
Case: Move Panel



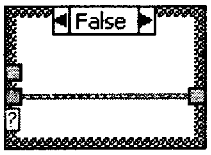
Case: Wait



Case: Exit



Case: False



2.1.6 List of SubVIs

Unload Module.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\Unload Module.vi

Register Module.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\Register Module.vi

Get Message.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\Get Message.vi

Send Message.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\Send Message.vi

CIC Set Interlock Setup Change.vi
C:\Controlatron\Development\Controlatron Interlock Control System\CIC Set Interlock Setup Change.vi

CIC Query Interlock State.vi
C:\Controlatron\Development\Controlatron Interlock Control System\CIC Query Interlock State.vi

CAS Set Interlock State.vi
C:\Controlatron\Development\Controlatron Acquisition System\CAS Set Interlock State.vi

CCC Set Interlock State.vi
C:\Controlatron\Development\Controlatron Control Center\CCC Set Interlock State.vi

PCS 01 001A Set Interlock State.vi
C:\Controlatron\Development\PCS 01 001A Power System\PCS 01 001A Set Interlock State.vi

CIC Variables.ctl
C:\Controlatron\Development\Controlatron Interlock Control System\CIC Variables.ctl

CIC Channel Setup.ctl
C:\Controlatron\Development\Controlatron Interlock Control System\CIC Channel Setup.ctl

CIC DAQ Get Interlock Status.vi
C:\Controlatron\Development\Controlatron Interlock Control System\CIC DAQ Get Interlock Status.vi

Respond To Message.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\Respond To Message.vi

Get Response Reference.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\Get Response Reference.vi

Read from Digital Port_With Error.vi
C:\Controlatron\Development\Controlatron Physical Drivers\DIO\Read from Digital Port_With Error.vi

Simple Error Handler.vi
C:\Program Files\National Instruments\LabVIEW 6\vi.lib\Utility\error.llb\Simple Error Handler.vi

CIC File Management.vi
C:\Controlatron\Development\Controlatron Interlock Control System\CIC File Management.vi

Get Screen Size.vi
C:\Controlatron\Development\Utilities\Winsys.llb\Get Screen Size.vi

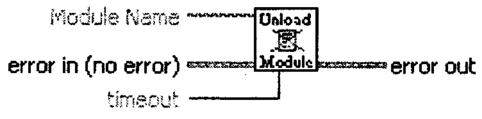
2.2 Unload Module.vi

This VI unloads the module specified by Module Name. If Module Name is not wired, the VI will unload the module it is running in. The process of unloading a module involves the following steps:
(1) The module's private queue is destroyed, thus preventing it from receiving messages.
(2) Execution of Module Name is terminated.
(3) Module Name is unloaded from memory.

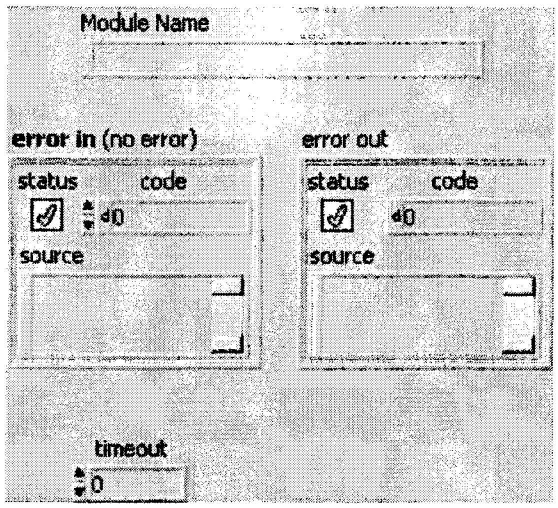
Since module execution may be aborted, the developer must be careful not to unload a module while the module is performing I/O processing or any other operations that should not be interrupted.

This VI will unload Module Name regardless of the incoming error condition.

2.2.1 Connector Pane



2.2.2 Front Panel



2.2.3 Controls and Indicators

Module Name. Name of the module to be unloaded.

error in (no error) The error in cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

status The status boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

code The code input identifies the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

source The source string describes the origin of the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

timeout

error out The error out cluster passes error or warning information out of a VI to be used by other VIs.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

TF **status** The status boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

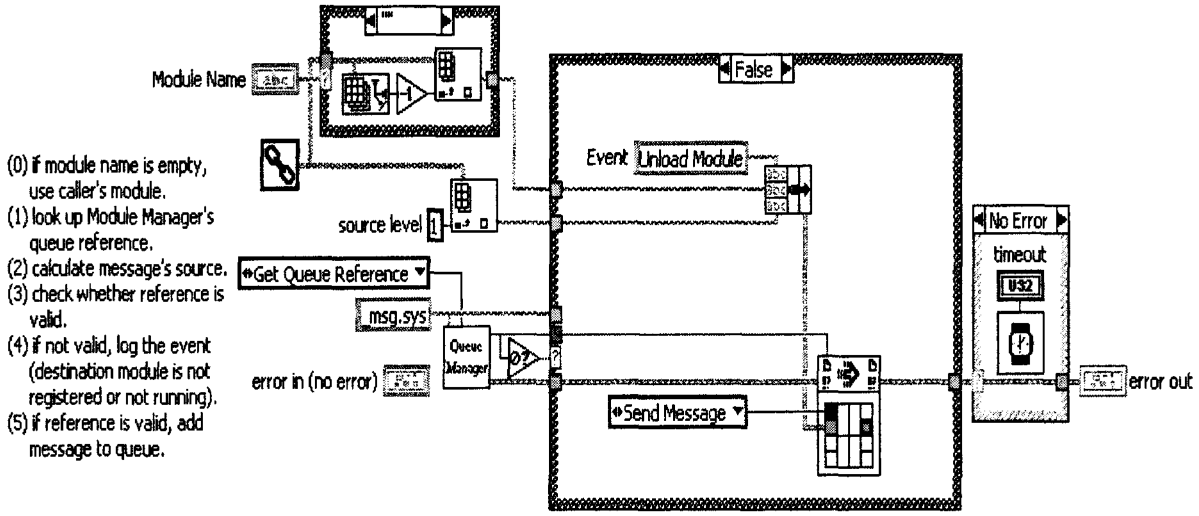
I32 **code** The code input identifies the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

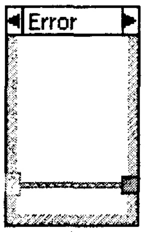
abc **source** The source string describes the origin of the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

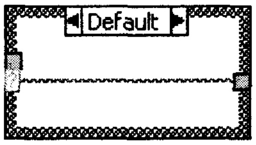
2.2.4 Block Diagram


Module Name
(0) if module name is empty, use caller's module.
(1) look up Module Manager's queue reference.
(2) calculate message's source.
(3) check whether reference is valid.
(4) if not valid, log the event (destination module is not registered or not running).
(5) if reference is valid, add message to queue.
source level
Get Queue Reference
msg.sys
Queue Manager
error in (no error)
False
Event
Unload Module
Send Message
No Error
timeout
U32
error out

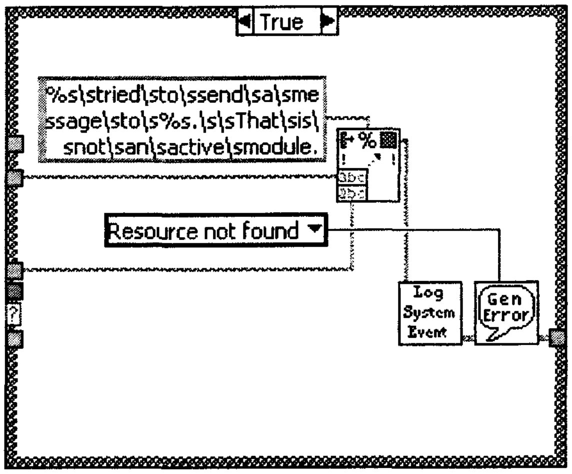
Case: Error



Case: Default



Case: True


True
%s\stried\sto\ssend\sa\sme
ssage\sto\s%s.\s\sThat\sis\
snot\san\sactive\smodule.
Resource not found
Log System Event
Gen Error

2.2.5 List of SubVIs

Queue Manager.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\Queue Manager.vi

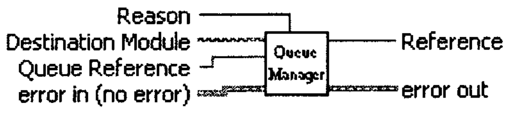
log system event.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\log system event.vi

Generate Error.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\Generate Error.vi

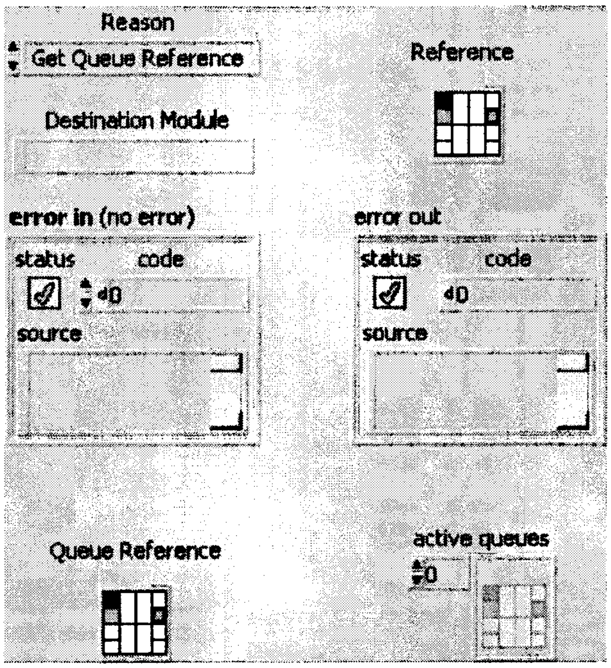
2.3 Queue Manager.vi

Reserved.

2.3.1 Connector Pane



2.3.2 Front Panel



2.3.3 Controls and Indicators

Destination Module

Reason

error in (no error) The error in cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

status The status boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

code The code input identifies the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

source The source string describes the origin of the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

active queues

Queue Reference 2

Queue Reference

Reference

error out The error out cluster passes error or warning information out of a VI to be used by other VIs.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

status The status boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

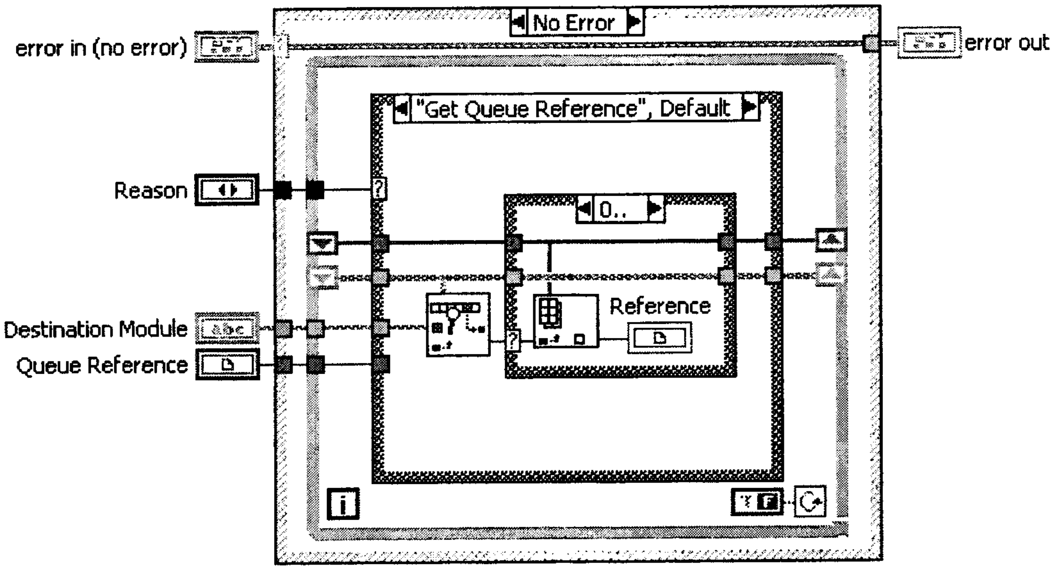
code The code input identifies the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

source The source string describes the origin of the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

2.3.4 Block Diagram



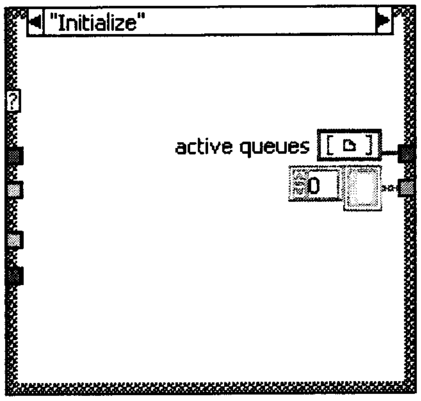
Initialize reason:
(1) clear queue names and references lists.

Create Queue reason:
(1) add module name and queue reference to the internal lists.

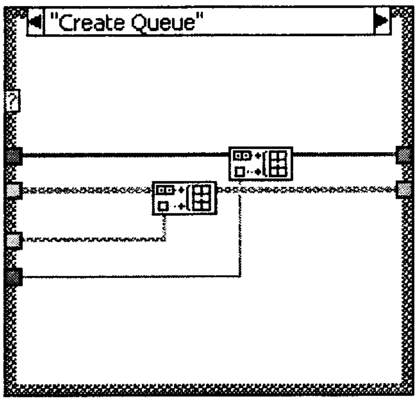
Remove Queue reason:
(1) look up module name in the manager lists.
(2) if module is not on the list, don't do anything.
(3) if module is on the list, destroy its queue (remove queue name and reference from the lists).

Get Queue reason:
(1) look up module name in the manager lists.
(2) if module is not on the list, don't do anything.
(3) if module is on the list, find its queue reference.
(4) return queue reference.

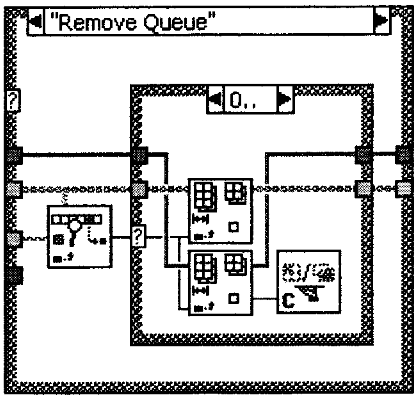
Case: Initialize



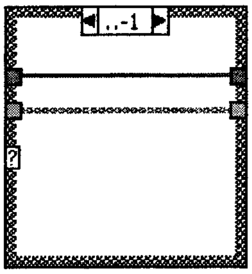
Case: Create Queue



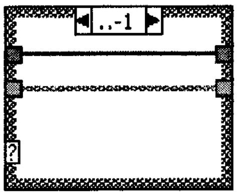
Case: Remove Queue



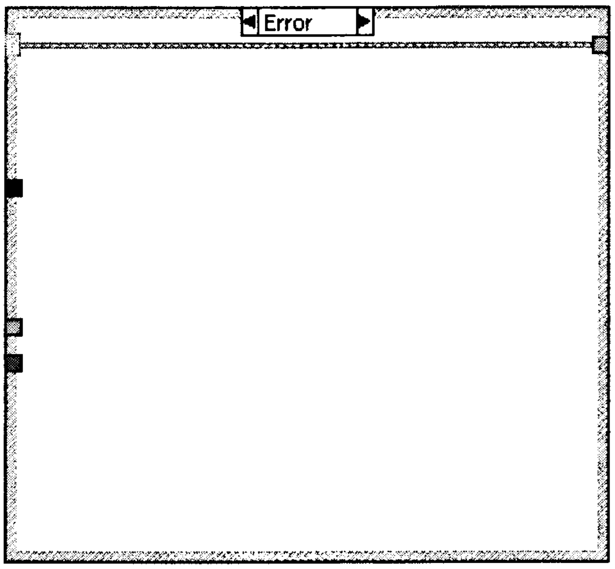
Case: -1



Case: -1



Case: Error



2.3.5 List of SubVIs

None

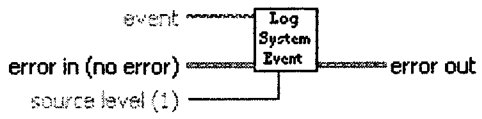
2.4 Log System Event.vi

This VI adds an event to the application's event log.

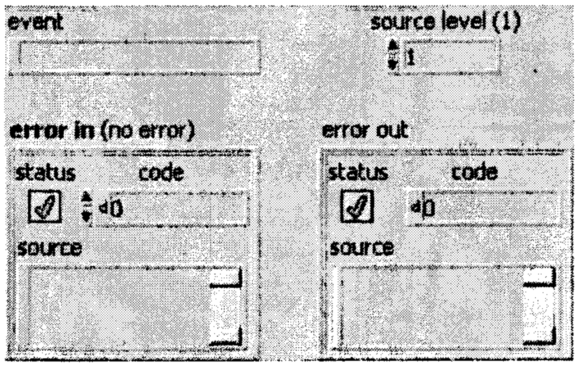
If there's an error condition in the error in cluster, this VI will add the error information to the event log. If there's no error condition, the VI will add the information in the event string to the event log.

If there's no error condition and event is empty (or unwired) this VI will do nothing.

2.4.1 Connector Pane



2.4.2 Front Panel



2.4.3 Controls and Indicators

Error in (no error) The error in cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

> **status** The status boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.
>
> The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.
>
> **code** The code input identifies the error or warning.
>
> The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.
>
> **source** The source string describes the origin of the error or warning.
>
> The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

event

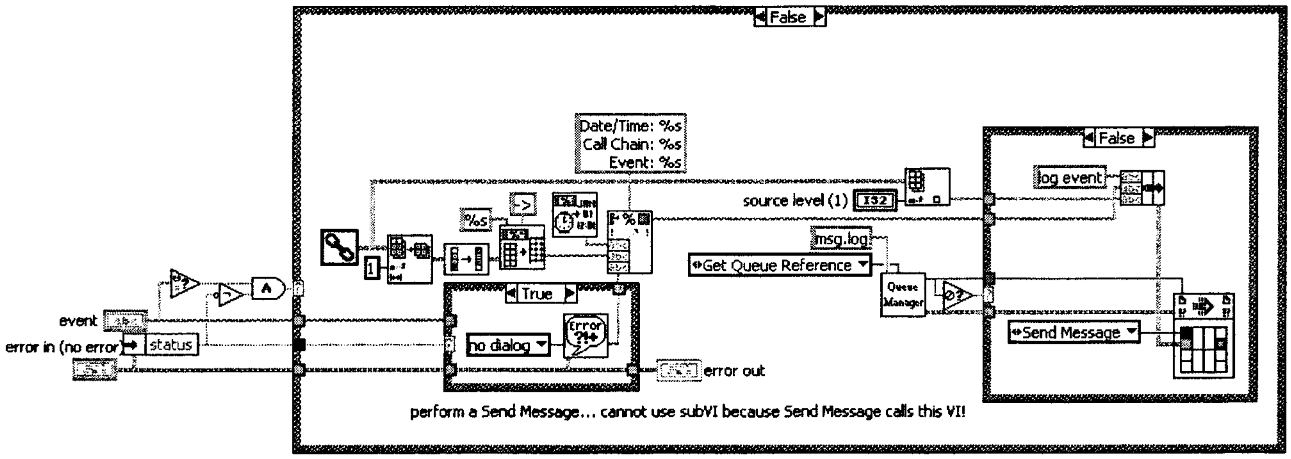
source level (1)

error out The error out cluster passes error or warning information out of a VI to be used by other VIs.

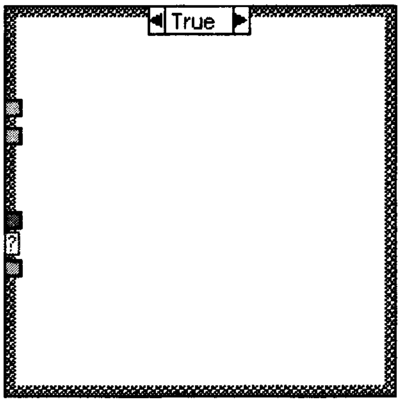
The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

> **status** The status boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.
>
> The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.
>
> **code** The code input identifies the error or warning.
>
> The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.
>
> **source** The source string describes the origin of the error or warning.
>
> The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

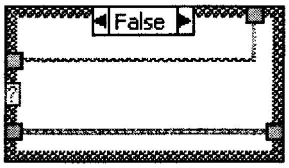
2.4.4 Block Diagram



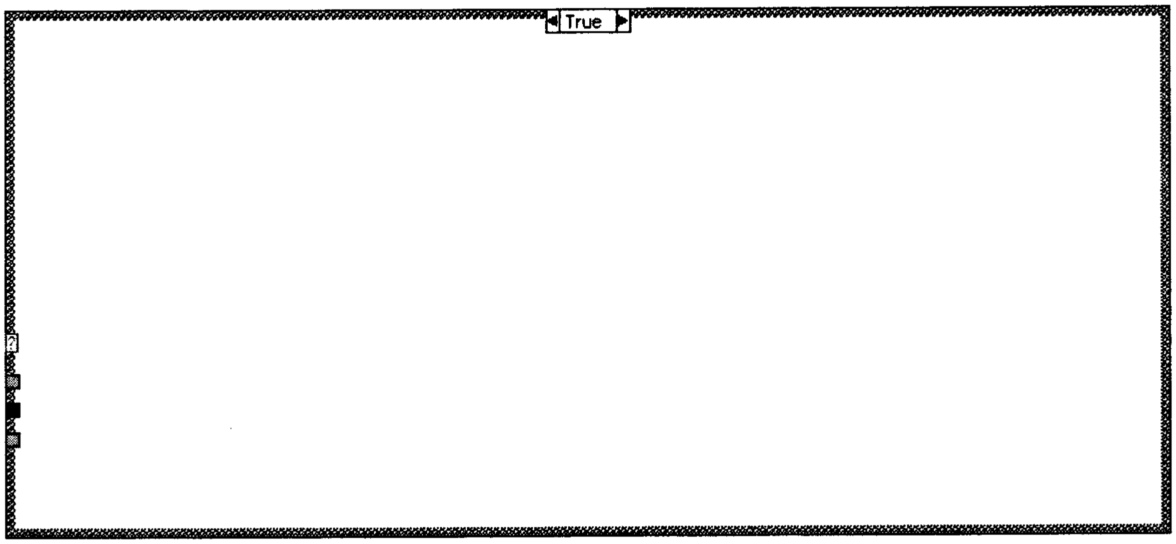
Case: True



Case: False



Case: True



2.4.5 List of SubVIs

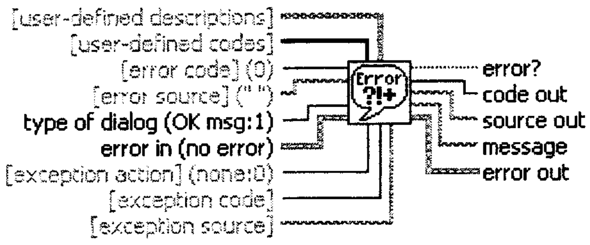
General Error Handler.vi
C:\Program Files\National Instruments\LabVIEW 6\vi.lib\Utility\error.llb\General Error Handler.vi

Queue Manager.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\Queue Manager.vi

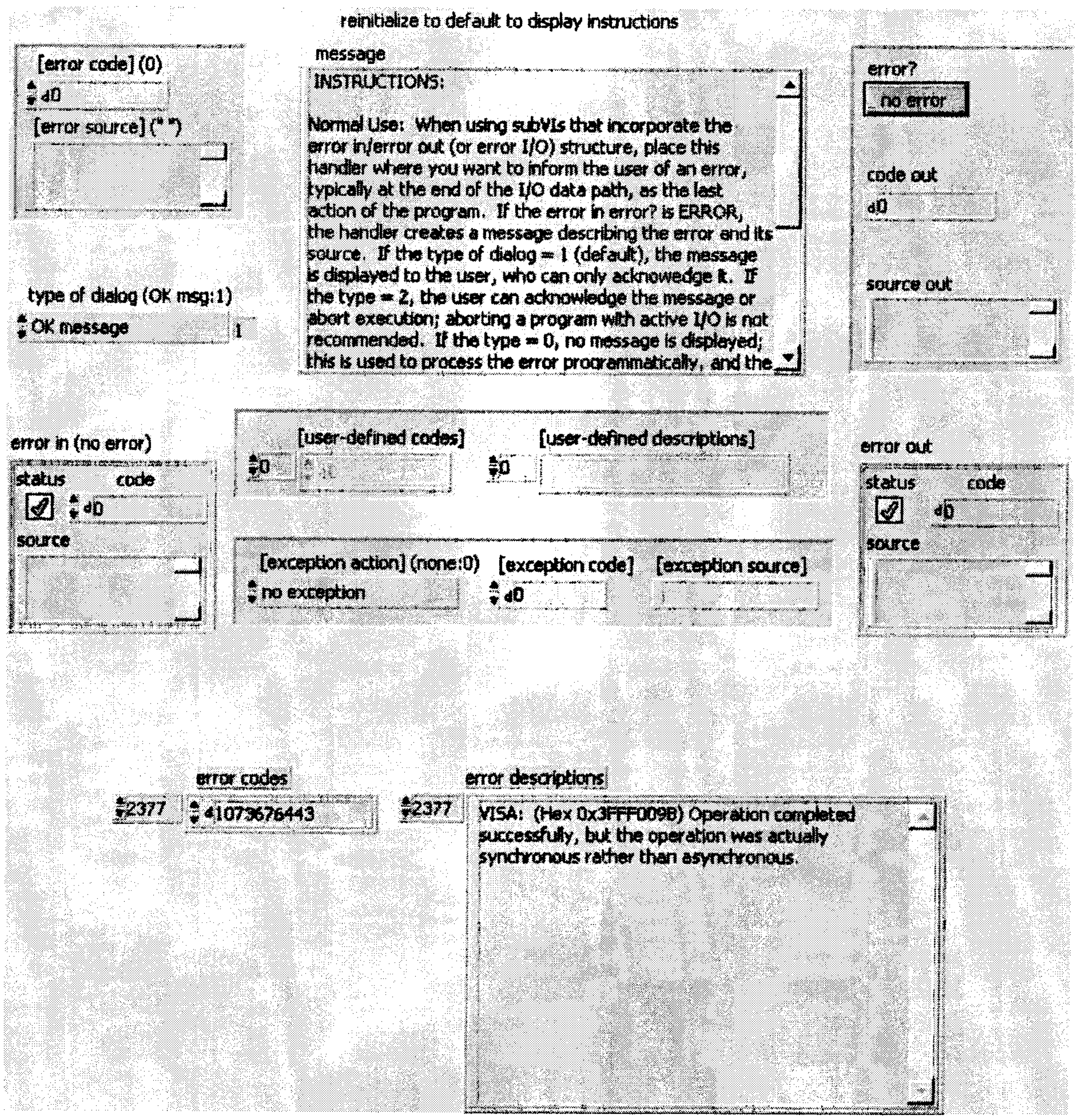
2.5 General Error Handler.vi

Determines whether an error has occurred. If an error has occurred, this VI creates a description of the error and optionally displays a dialog box.

2.5.1 Connector Pane



2.5.2 Front Panel



INSTRUCTIONS:

Normal Use: When using subVIs that incorporate the error in/error out (or error I/O) structure, place this handler where you want to inform the user of an error, typically at the end of the I/O data path, as the last action of the program. If the error in error? is ERROR, the handler creates a message describing the error and its source. If the type of dialog = 1 (default), the message is displayed to the user, who can only acknowedge it. If the type = 2, the user can acknowledge the message or abort execution; aborting a program with active I/O is not recommended. If the type = 0, no message is displayed; this is used to process the error programmatically, and the

VISA: (Hex 0x3FFF009B) Operation completed successfully, but the operation was actually synchronous rather than asynchronous.

2.5.3 Controls and Indicators

U16 **type of dialog (OK msg:1) type of dialog** determines what type of dialog box to display, if any.

I32 **[exception code] exception code** is the error code that you want to treat as an exception. By default, it is 0.

abc **[exception source] exception source** is the error message that you want to use to test for an exception. By default, it is an empty string.

[error source] (" ") error source is an optional string you can use to describe the source of **error code**.

[error code] (0) error code is a *numeric error code.*

[exception action] (none:0) exception action is a way *for you to create exceptions to* error handling.

[user-defined codes] user-defined codes is an array of the numeric error codes you define in your VIs.

user-defined codes is an array of the numeric *error codes you define in your* VIs.

[user-defined descriptions] user-defined descriptions is *an array of descriptions of user-*defined codes.

user-defined descriptions is an array of descriptions of user-defined codes.

error descriptions

error codes

prompts

Error %d occurred at %s

Warning %d occurred at %s

Possible reasons: %s

Error not listed

GetCommError x%lx

Continue

Stop

an unidentified location

No Error

error in (no error) The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

status The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

code The **code** numeric identifies the error or warning.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

source The **source** string describes the origin of the error or warning.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

code out code out is the error code indicated by the **error in** or **error code.**

source out source out indicates the source of the error.

error? error? displays if an error was found. If this VI finds an error, it sets the parameters in the error cluster.

message message describes the error code that occurred, the source of the error, and a description of the error.

error out The **error out** cluster passes error or warning information out of a VI to be used by other VIs.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

status The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

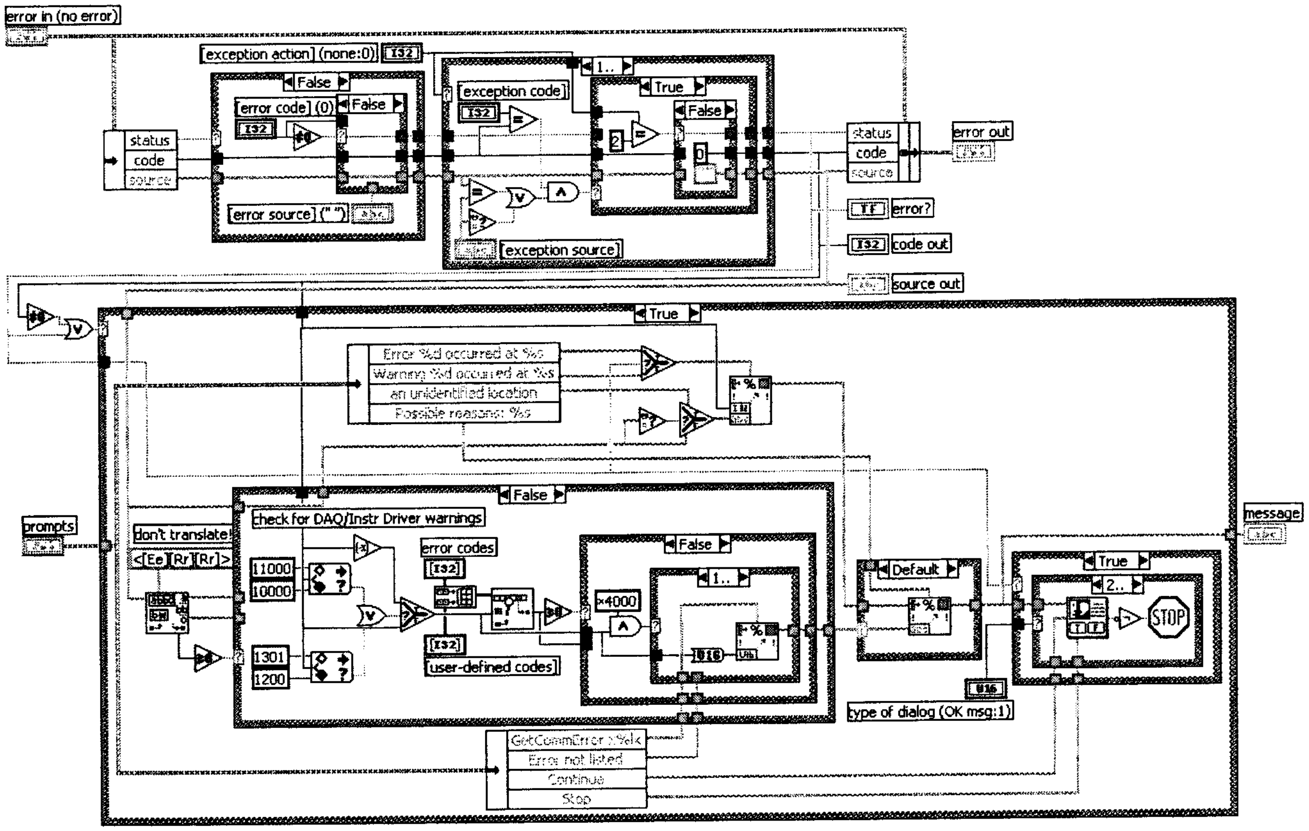
code The **code** numeric identifies the error or warning.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

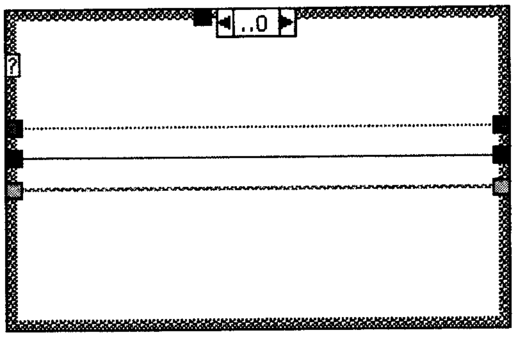
source The **source** string describes the origin of the error or warning.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

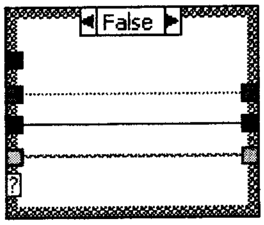
2.5.4 Block Diagram



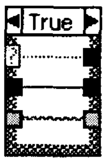
Case: 0



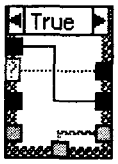
Case: False



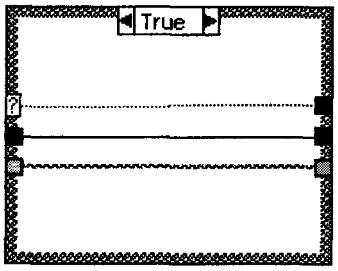
Case: True



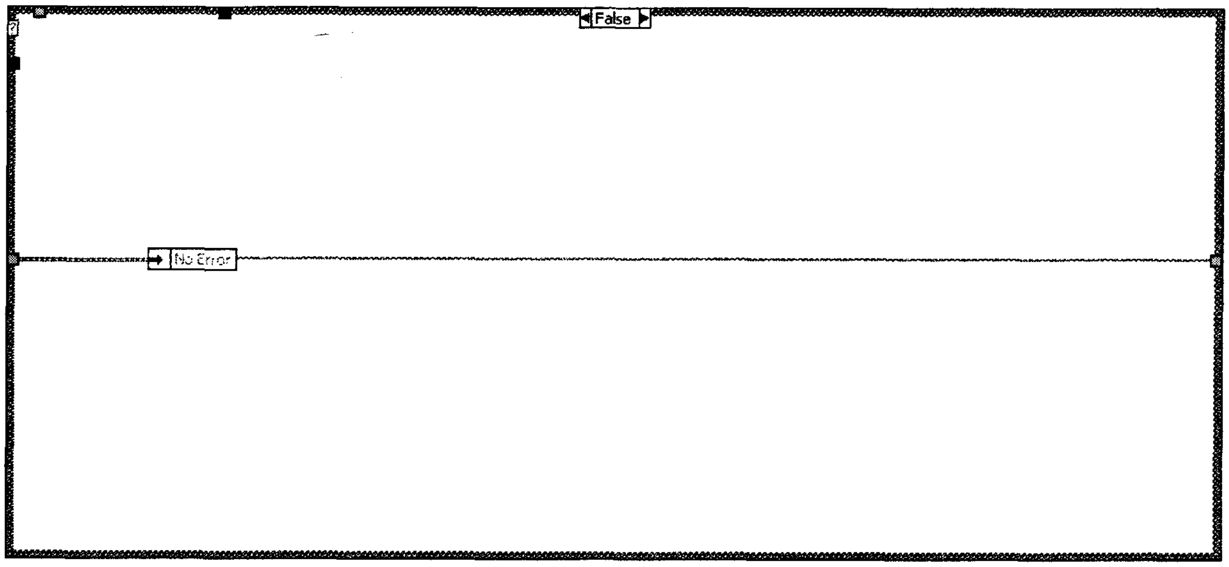
Case: True



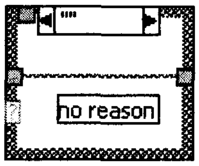
Case: True



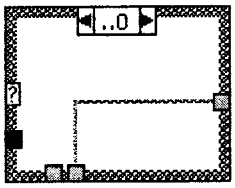
Case: False



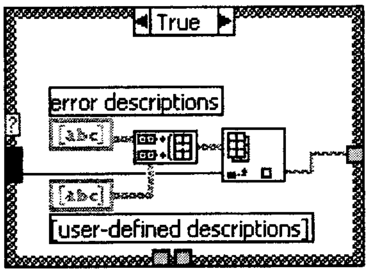
Case: ""



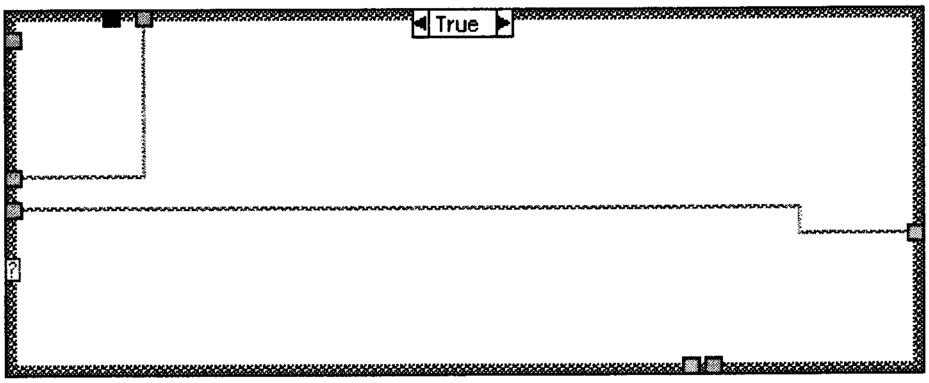
Case: 0



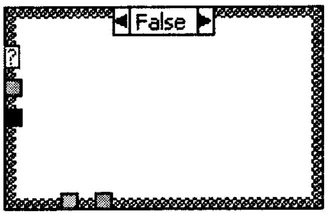
Case: True



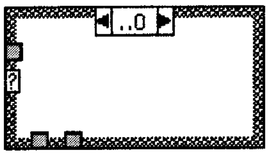
Case: True



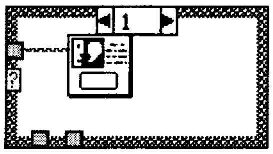
Case: False



Case: 0



Case: 1



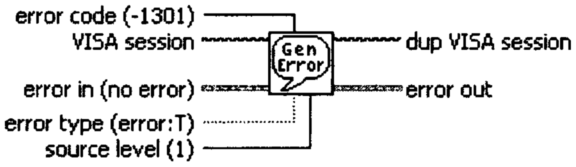
2.5.5 List of SubVIs

None

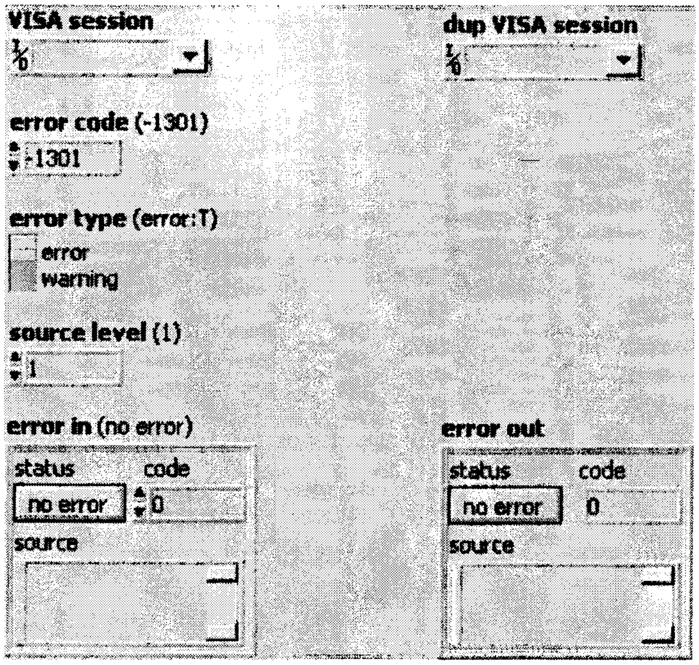
2.6 Generate Error.vi

This VI generates a user-defined error in the "error out" cluster.

2.6.1 Connector Pane



2.6.2 Front Panel



2.6.3 Controls and Indicators

VISA session

error in (no error)

TF **status** status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

I32 **code** code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

abc **source** source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

TF **error type (error:T)** Selects whether to generate an error or a warning:

error = asserts an "error out" 'status' of true, which is system critical and will disable all polling of instrumentation until it is cleared.

warning = asserts an "error out" 'status' of false, which is not system critical and will not disable the polling of instrumentation.

I32 **error code (-1301)** The error code that will be placed in 'code' of "error out".

Several regions of the error code "band" (an I32 number) are allocated for generating errors external to LabVIEW.

Instrument Driver Error Codes:
-1210 = Parameter out of range
-1223 = Instrument identification query failed
-1236 = Error Interpreting instrument response
-1300 = Instrument-specific error
-1301 to -1399 = is used for any instrument specific errors. This band of errors may be assigned developer-defined error messages using the "General Error Handler.vi". The error code -1300 should be avoided, because it is a generic instrument driver error and a specific error message cannot be generated for it.

User-Defined Error Codes:
A region from 5000 to 9999 is set aside for any user-defined errors. This band of errors may be assigned user-defined error messages using the "General Error Handler.vi".

IMPORTANT:
Because it is possible to assign error codes that overlap with other previously defined error codes (i.e. several instruments may use the -1302 error code which may mean different things to those instruments), it is important to use several techniques to avoid this.

1) Instead of using the "General Error Handler.vi", instead use "BBT Error Handler.vi". This error handler only generates an error message for instrument-specific and user-defined errors if the incoming 'source' in "error in" contains a specific string, which is set by the developer. For example, "BBT Error Handler.vi" would look for an instrument prefix in an incoming error for an instrument driver. It is, therefore, important to follow naming conventions set for instrument drivers.

2) Make sure every instrument driver is bundled with its own "PREFIX Error Message.vi" which uses the "BBT Error Handler.vi" and decodes and displays its own error messages.

source level (1) Specifies the level of the "call chain" at which the error was generated.

Since the call chain includes all of the names of every VI from this VI to all others that call it, it is useful to exclude the name of this VI from the 'source' in "error out". To exclude the name of this VI, a 1 is entered. In order to exclude other branches of the error source, increment this control the number of branches to exclude.

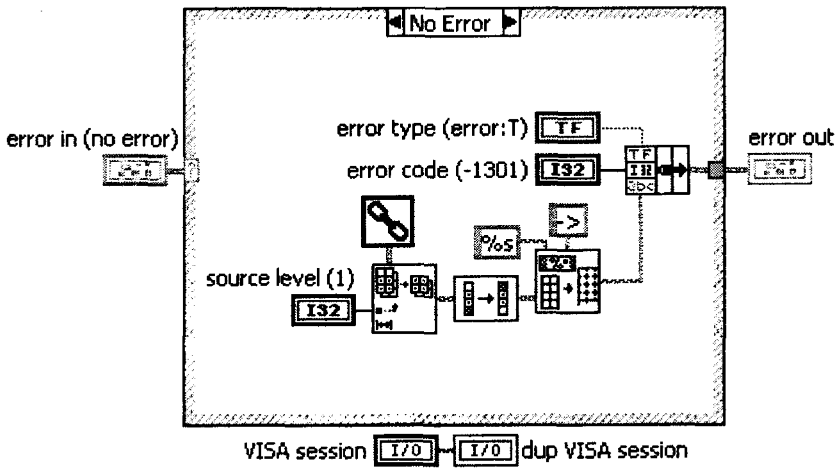
error out

- **status** status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.
- **code** code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.
- **source** source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

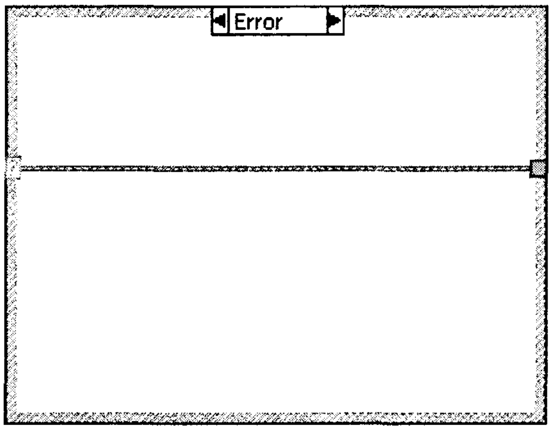
dup VISA session

2.6.4 Block Diagram

Case:No Error



Case: Error



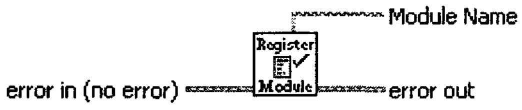
2.6.5 List of SubVIs

2.7 Register Module.vi

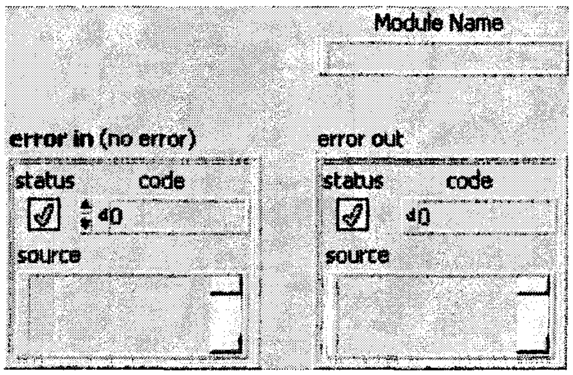
This VI registers the module that calls it and returns the module's name for future references. Any module that needs to receive messages must first register or else it will not be capable of receiving messages. Registration causes the framework to create a private message queue for the module.

If there is an incoming error, this VI will not register the module. It will only propagate the error information to error out.

2.7.1 Connector Pane



2.7.2 Front Panel



2.7.3 Controls and Indicators

error in (no error) The error in cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

- **status** The status boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

  The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

- **code** The code input identifies the error or warning.

  The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

- **source** The source string describes the origin of the error or warning.

  The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

Module Name. Name of the module that was registered.

error out The error out cluster passes error or warning information out of a VI to be used by other VIs.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

- **status** The status boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

  The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

- **code** The code input identifies the error or warning.

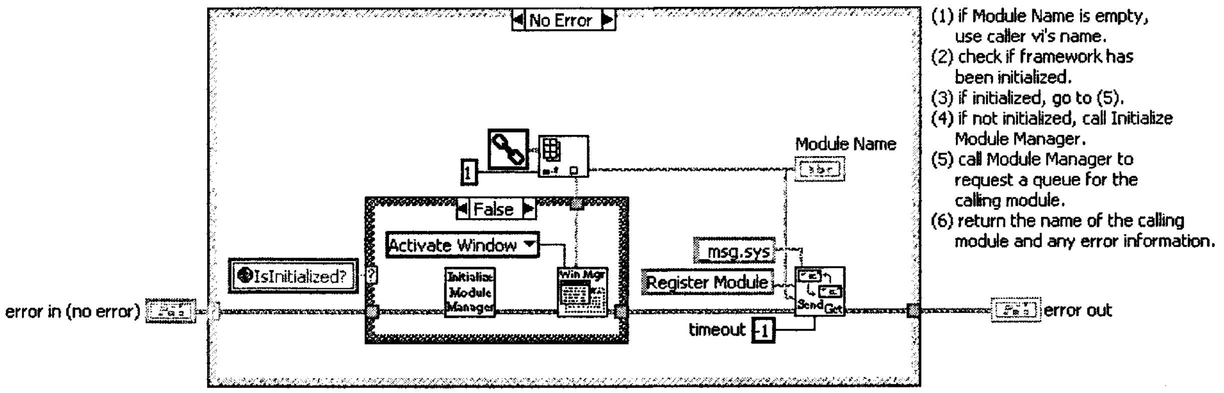
  The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

- **source** The source string describes the origin of the error or warning.

  The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

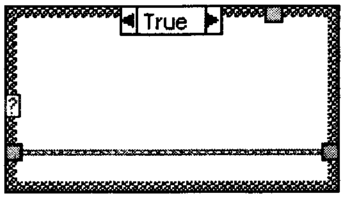
2.7.4 Block Diagram

Case: No Error

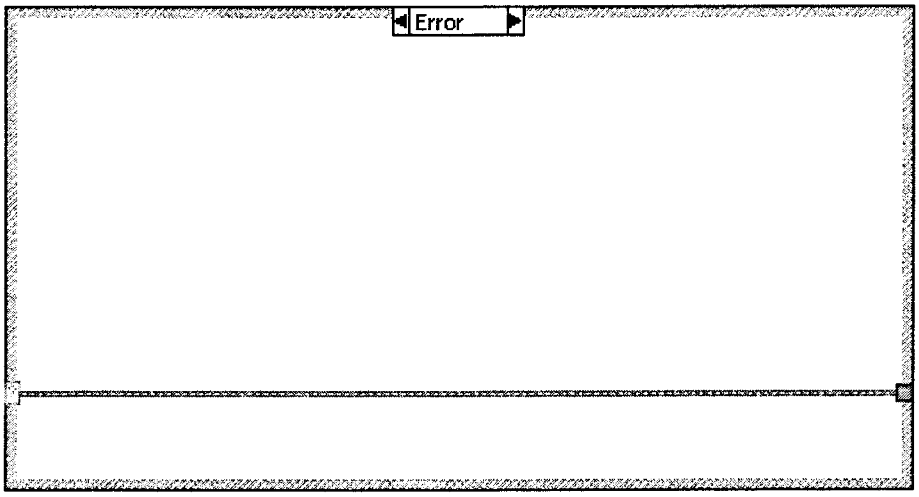


(1) if Module Name is empty, use caller vi's name.
(2) check if framework has been initialized.
(3) if initialized, go to (5).
(4) if not initialized, call Initialize Module Manager.
(5) call Module Manager to request a queue for the calling module.
(6) return the name of the calling module and any error information.

Case: True



Case: Error



2.7.5 List of SubVIs

MA Definitions.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\MA Definitions.vi

Initialize Module Manager.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\Initialize Module Manager.vi

Send Message-Get Response.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\Send Message-Get Response.vi

Set Windows State.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\Set Windows State.vi

windows state.ctl
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\windows state.ctl

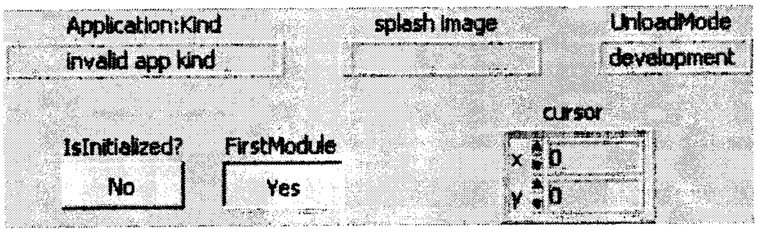
2.8 MA Definitions.vi

This VI defines fundamental data types used by the Module Architecture (MA) VIs. It is not intended to be called directly by the user.

2.8.1 Connector Pane

MA
Type
Defs.

2.8.2 Front Panel


Application:Kind: invalid app kind
splash image:
UnloadMode: development
cursor: x 0, y 0
IsInitialized?: No
FirstModule: Yes

2.8.3 Controls and Indicators

IsInitialized?

splash image

FirstModule

cursor

- **x**
- **y**

Application:Kind

UnloadMode

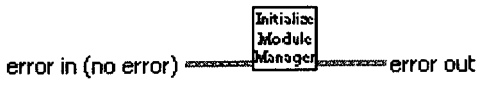
2.8.4 Block Diagram

2.8.5 List of SubVIs

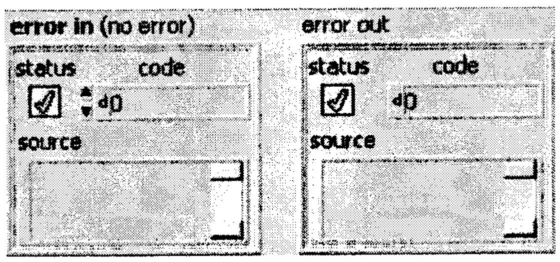
2.9 Initialize Module Manager.vi

Reserved.

2.9.1 Connector Pane



2.9.2 Front Panel



2.9.3 Controls and Indicators

error in (no error) The error in cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

> **status** The status boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.
>
> The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.
>
> **code** The code input identifies the error or warning.
>
> The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.
>
> **source** The source string describes the origin of the error or warning.
>
> The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

Error out The error out cluster passes error or warning information out of a VI to be used by other VIs.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

status The status boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

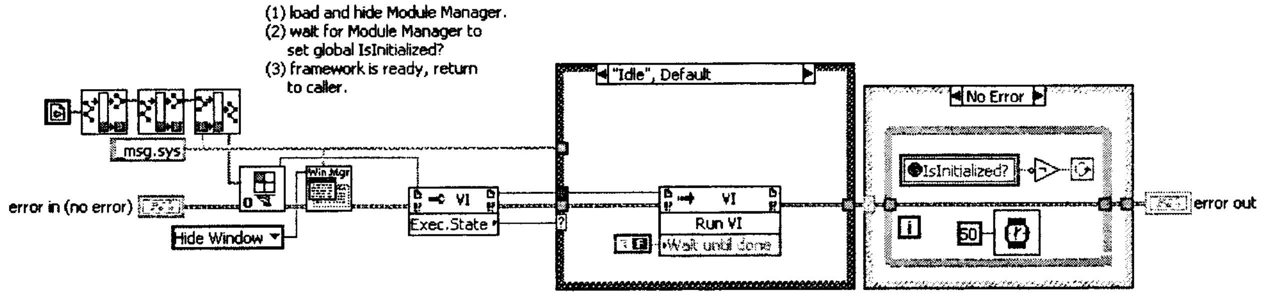
code The code input identifies the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

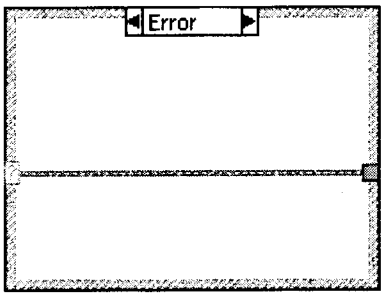
source The source string describes the origin of the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

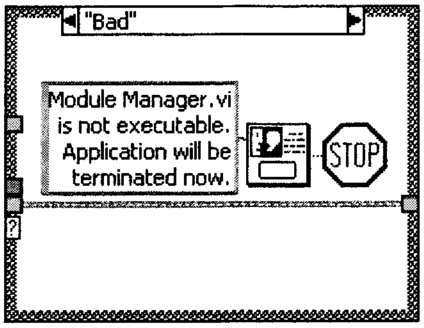
2.9.4 Block Diagram


(1) load and hide Module Manager.
(2) wait for Module Manager to set global IsInitialized?
(3) framework is ready, return to caller.
msg.sys
error in (no error)
Hide Window
Win Mgr
VI
Exec.State
"Idle", Default
Run VI
Wait until done
No Error
IsInitialized?
50
error out

Case: Error

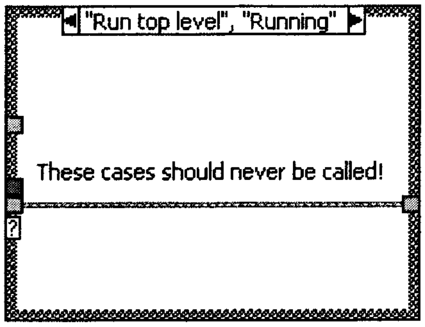


Case: Bad



"Bad"
Module Manager.vi is not executable. Application will be terminated now.
STOP
?

Case: Run top level, Running



"Run top level", "Running"
These cases should never be called!
?

2.9.5 List of SubVIs

MA Definitions.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\MA Definitions.vi

Set Windows State.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\Set Windows State.vi

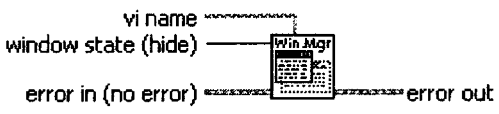
windows state.ctl
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\windows state.ctl

2.10 Set Windows State.vi

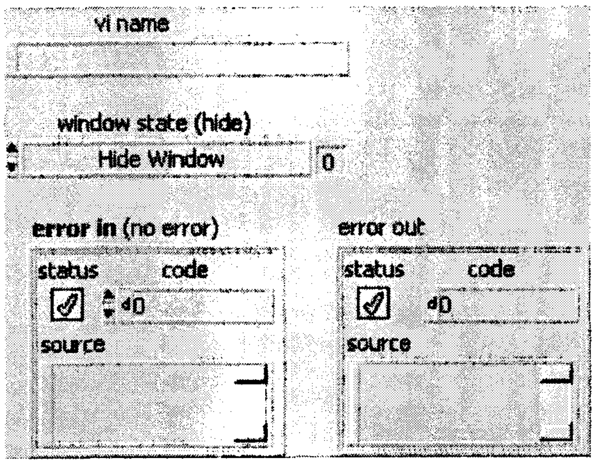
This VI calls the Windows operating system to control the state of the window specified by vi name.

This VI will not execute if there is an incoming error in the error in cluster.

2.10.1 Connector Pane



2.10.2 Front Panel



2.10.3 Controls and Indicators

error in (no error) The error in cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

status The status boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

code The code input identifies the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

source The source string describes the origin of the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

I32 **window state (hide)** Specifies how the window is to be shown. Window state can be one of the following:

HIDE - Hides the window and activates another window.

MAXIMIZE - Maximizes the specified window.

MINIMIZE - Minimizes the specified window and activates the next top-level window in the Z order.

RESTORE - Activates and displays the window. If the window is minimized or maximized, Windows restores it to its original size and position. An application should specify this flag when restoring a minimized window.

SHOW - Activates the window and displays it in its current size and position.

SHOWDEFAULT - Sets the state based on the startup information given by the program that started the application. An application should use this setting to set the initial state of its main window.

SHOWMAXIMIZED - Activates the window and displays it as a maximized window.

SHOWMINIMIZED - Activates the window and displays it as a minimized window.

SHOWMINNOACTIVE - Displays the window as a minimized window. The active window remains active.

SHOWNA - Displays the window in its CURRENT state. The active window remains active.

SHOWNOACTIVATE - Displays a window in its most recent size and position. The active window remains active.

SHOWNORMAL - Activates and displays a window. If the window is minimized or maximized, Windows restores it to its original size and position. An application should specify this setting when displaying the window for the first time.

abc **vi name**

error out The error out cluster passes error or warning information out of a VI to be used by other VIs.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

TF **status** The status boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

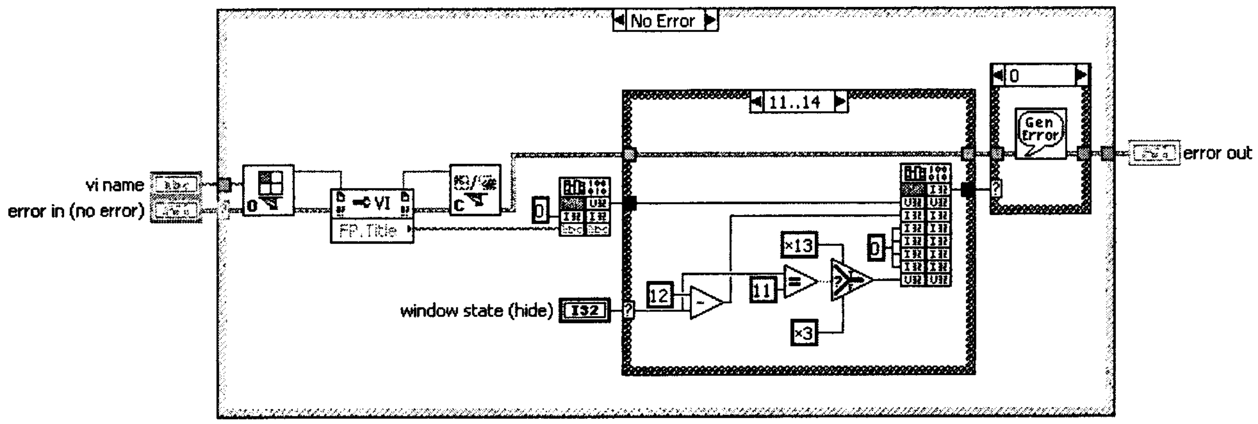
I32 **code** The code input identifies the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

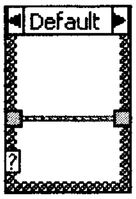
source The source string describes the origin of the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

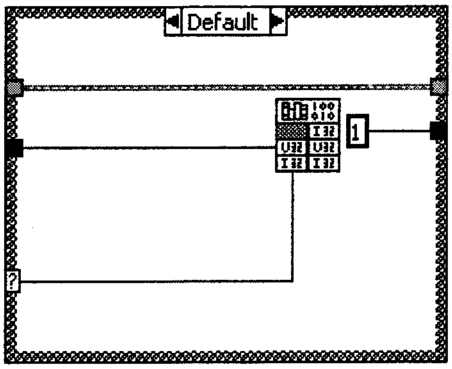
2.10.4 Block Diagram



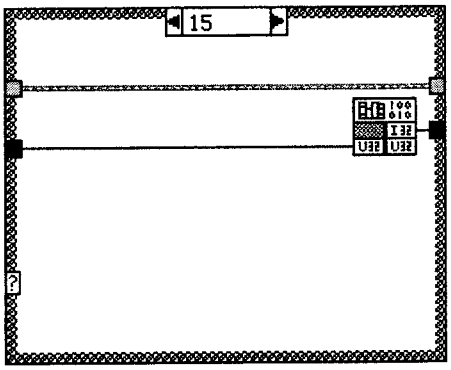
Case: Default



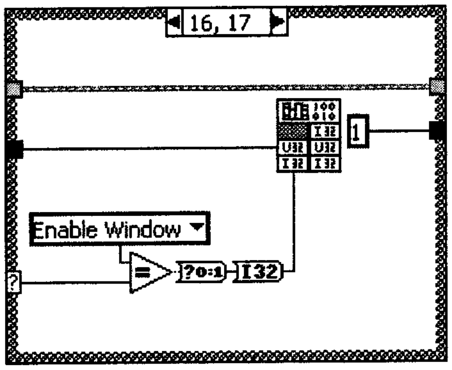
Case: Default



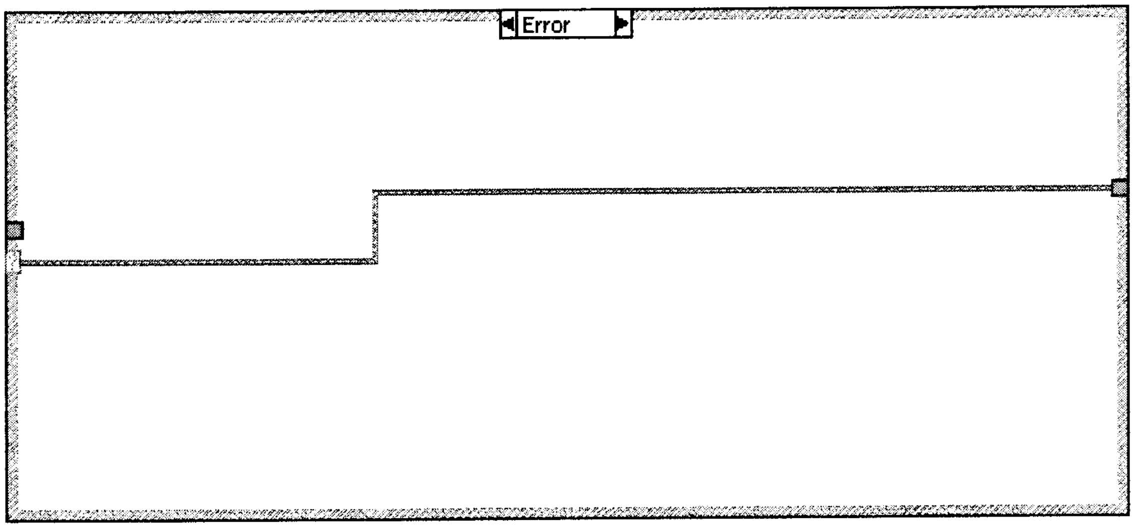
Case: 15



Case: 16,17



Case: Error



2.10.5 List of SubVIs

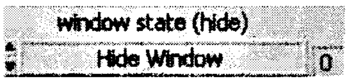
Generate Error.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\Generate Error.vi

windows state.ctl
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\windows state.ctl

2.11 Windows State.ctl

2.11.1 Connector Pane

2.11.2 Front Panel


window state (hide)
Hide Window
0

2.11.3 Controls and Indicators

I32 **window state (hide)** Specifies how the window is to be shown. Window state can be one of the following:

HIDE - Hides the window and activates another window.

MAXIMIZE - Maximizes the specified window.

MINIMIZE - Minimizes the specified window and activates the next top-level window in the Z order.

RESTORE - Activates and displays the window. If the window is minimized or maximized, Windows restores it to its original size and position. An application should specify this flag when restoring a minimized window.

SHOW - Activates the window and displays it in its current size and position.

SHOWDEFAULT - Sets the state based on the startup information given by the program that started the application. An application should use this setting to set the initial state of its main window.

SHOWMAXIMIZED - Activates the window and displays it as a maximized window.

SHOWMINIMIZED - Activates the window and displays it as a minimized window.

SHOWMINNOACTIVE - Displays the window as a minimized window. The active window remains active.

SHOWNA - Displays the window in its CURRENT state. The active window remains active.

SHOWNOACTIVATE - Displays a window in its most recent size and position. The active window remains active.

SHOWNORMAL - Activates and displays a window. If the window is minimized or maximized, Windows restores it to its original size and position. An application should specify this setting when displaying the window for the first time.

2.11.4 Block Diagram

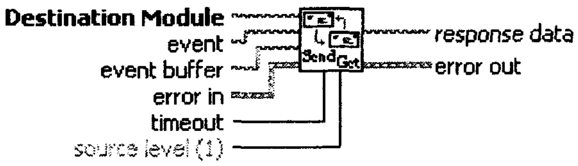
2.11.5 List of SubVIs

2.12 Send Message-Get Response.vi

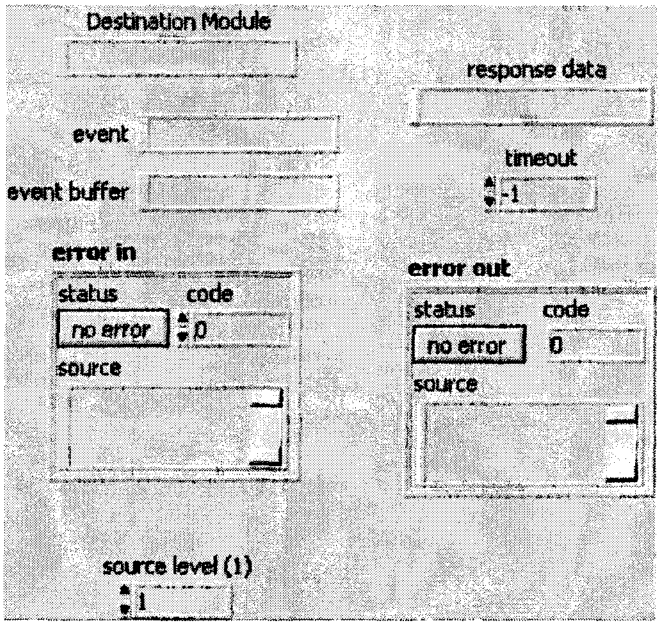
This VI delivers a message to Destination Module and then waits for the module's response. It will return control to the calling VI only after it receives the response. However, if the response is not received within timeout milliseconds, this VI will stop waiting and will return an error. Timeout defaults to infinite milliseconds if left unwired.

If this VI times out, error out will show a timeout error. In all other circumstances, error out will reflect the outcome of the process called. This VI will not execute if there is an error in error in.

2.12.1 Connector Pane



2.12.2 Front Panel



2.12.3 Controls and Indicators

error in error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

- **status** status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

- **code** code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

- **source** source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

timeout Maximum amount of time, expressed in milliseconds, to wait for the message response. A value of (-1) disables the timeout functionality (i.e., the VI will wait forever).

Destination Module Application module that will receive and handle the specified message.

event String that describes a task or event.

event buffer Data needed to perform the task specified by the event string.

source level (1)

error out error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

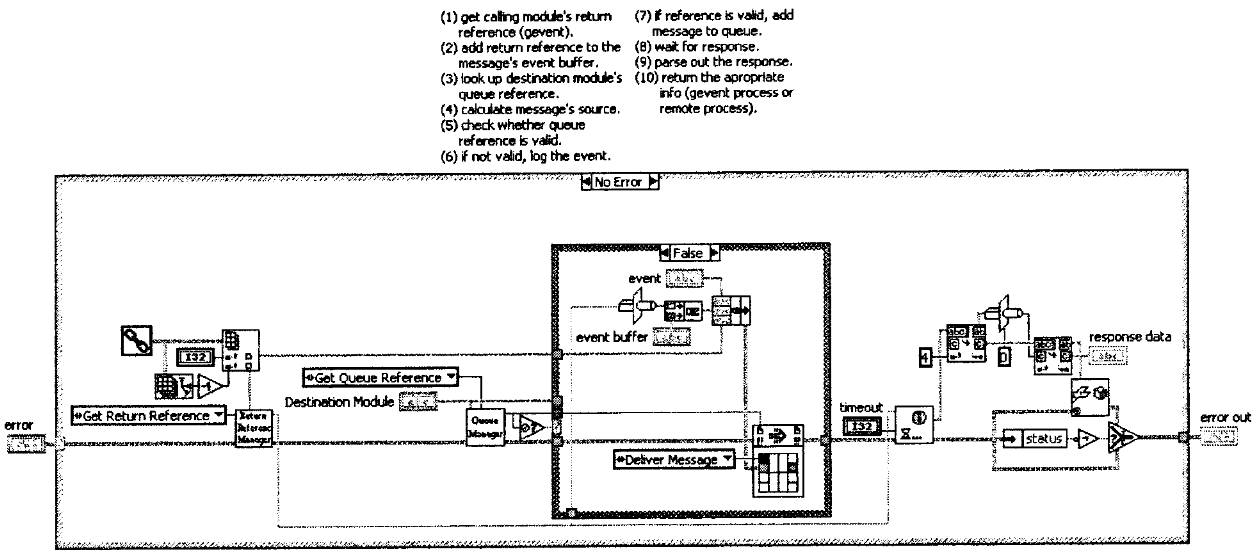
- **status** status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

- **code** code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

- **source** source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

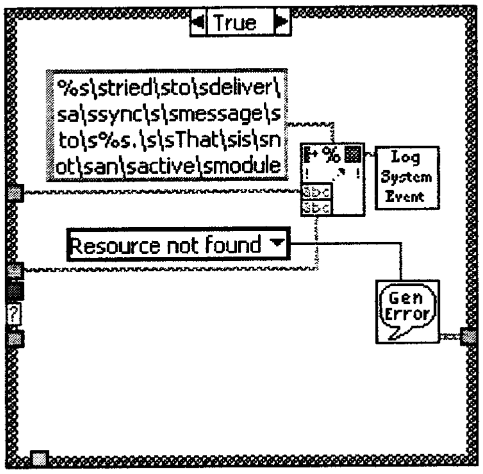
response data Data generated in response to the synchronous message.

2.12.4 Block Diagram



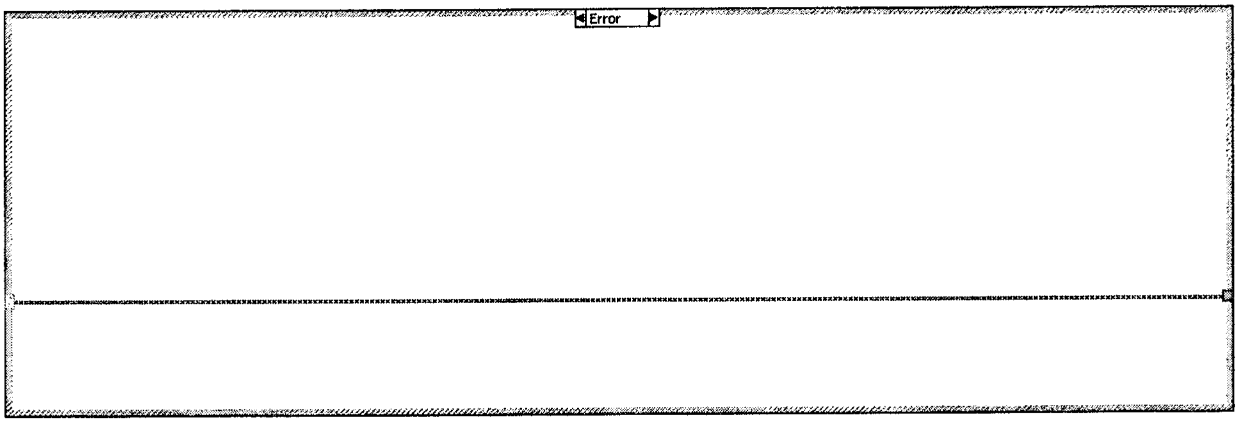
(1) get calling module's return reference (gevent).
(2) add return reference to the message's event buffer.
(3) look up destination module's queue reference.
(4) calculate message's source.
(5) check whether queue reference is valid.
(6) if not valid, log the event.
(7) if reference is valid, add message to queue.
(8) wait for response.
(9) parse out the response.
(10) return the apropriate info (gevent process or remote process).

Case: True



True
%s\stried\sto\sdeliver\sa\ssync\s\smessage\sto\s%s.\s\sThat\sis\snot\san\sactive\smodule
Log System Event
Resource not found
Gen Error

Case: Error



2.12.5 List of SubVIs

Return Reference Manager.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\Return Reference Manager.vi

Queue Manager.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\Queue Manager.vi

log system event.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\log system event.vi

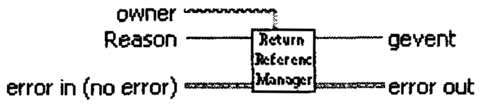
wait on gevent.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\wait on gevent.vi

Generate Error.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\Generate Error.vi

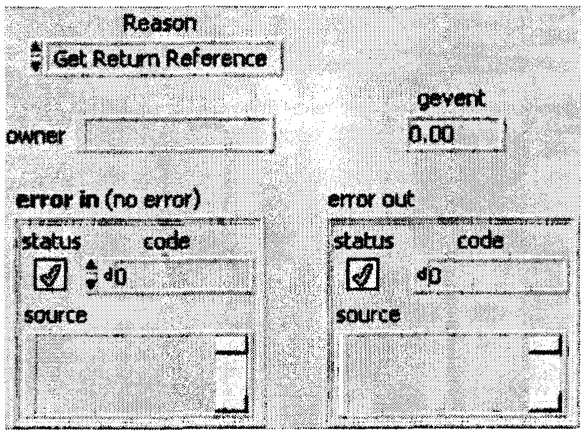
2.13 Return Reference Manager.vi

Reserved.

2.13.1 Connector Pane



2.13.2 Front Panel



2.13.3 Controls and Indicators

Reason

error in (no error) The error in cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

status The status boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

code The code input identifies the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

source The source string describes the origin of the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

owner

gevent

error out The error out cluster passes error or warning information out of a VI to be used by other VIs.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

status The status boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

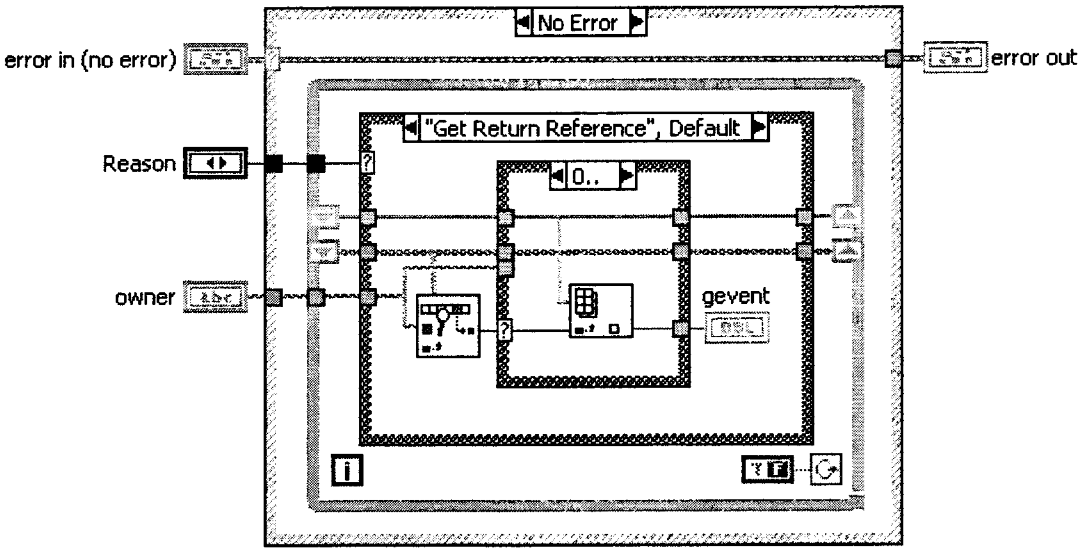
code The code input identifies the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

source The source string describes the origin of the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

2.13.4 Block Diagram



Initialize reason:
Clear owner and reference lists.

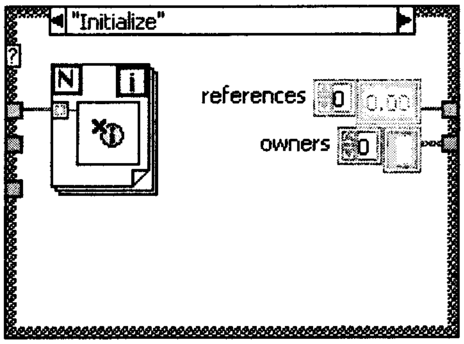
Create Reference reason:
(1) look up owner in the manager lists.
(2) if owner is not on the list, add specified owner and new reference to the internal lists.

Get Reference reason:
(1) look up owner in the manager lists.
(2) if owner is not on the list, create new reference and owner element.
(3) if owner is on the list, return its reference.

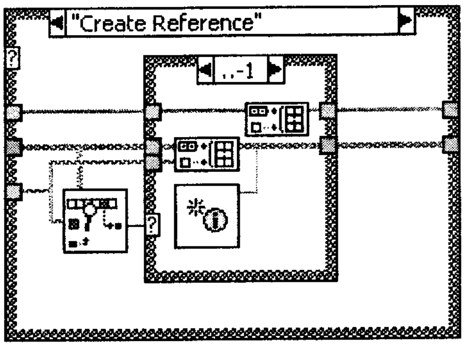
Remove Reference reason:
(1) look up owner in the manager lists.
(2) if owner is not on the list, don't do anything.
(3) if owner is on the list, destroy its reference (remove owner and reference from the lists).

Release Reference reason is obsolete.

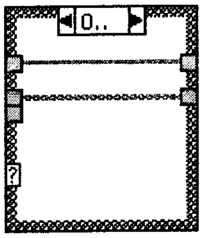
Case: Initialize



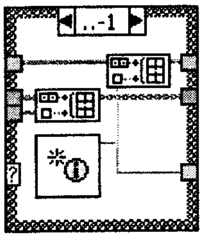
Case: Create Reference



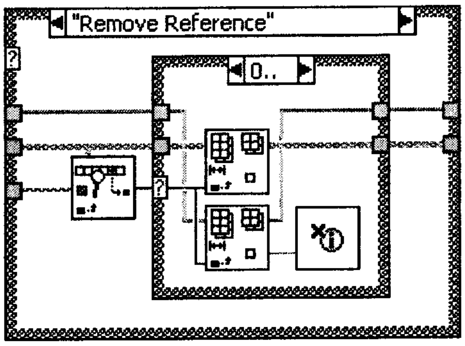
Case: 0



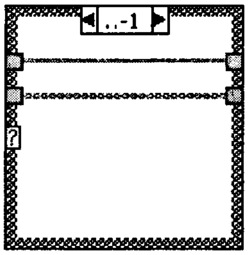
Case: -1



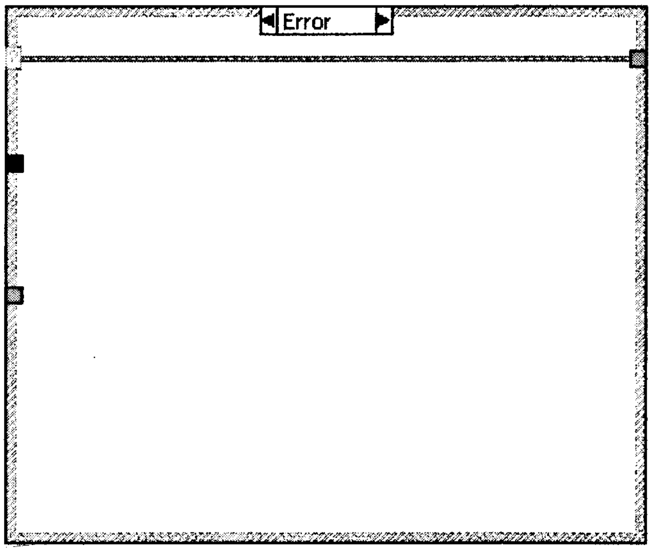
Case: Remove Reference



Case: -1



Case: Error



2.13.5 List of SubVIs

create gevent.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\create gevent.vi

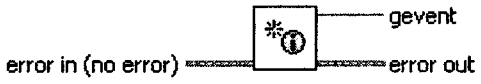
destroy gevent.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\destroy gevent.vi

2.14 create gevent.vi

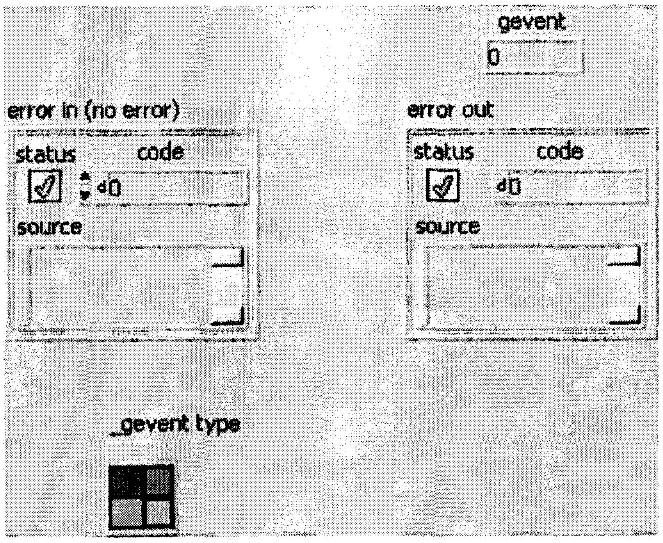
Creates a new gevent and returns a refnum that you can use when calling other gevent VIs.

This VI will not work if error in specifies an error condition.

Connector Pane



2.14.1 Front Panel



2.14.2 Controls and Indicators

error in (no error) The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

- **status** The status boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

  The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

- **code** The code input identifies the error or warning.

  The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

- **source** The source string describes the origin of the error or warning.

  The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

_gevent type

error out The error out cluster passes error or warning information out of a VI to be used by other VIs.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

status The status boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

code The code input identifies the error or warning.

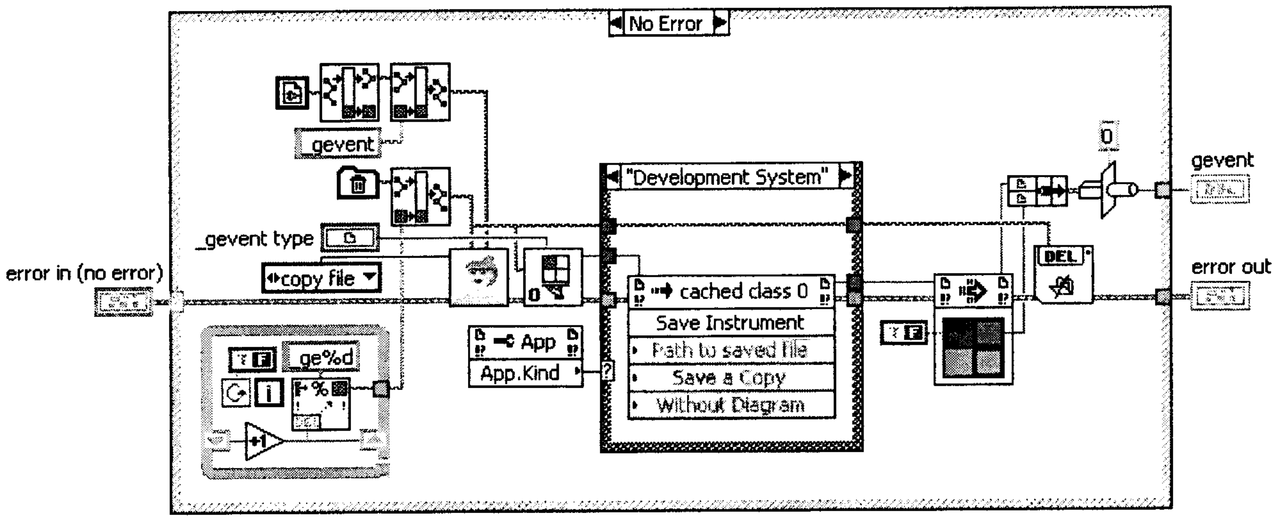
The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

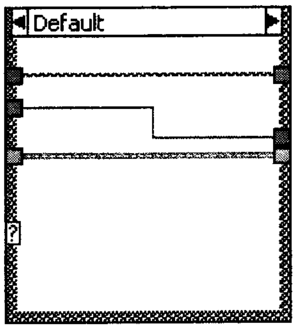
source The source string describes the origin of the error or warning.

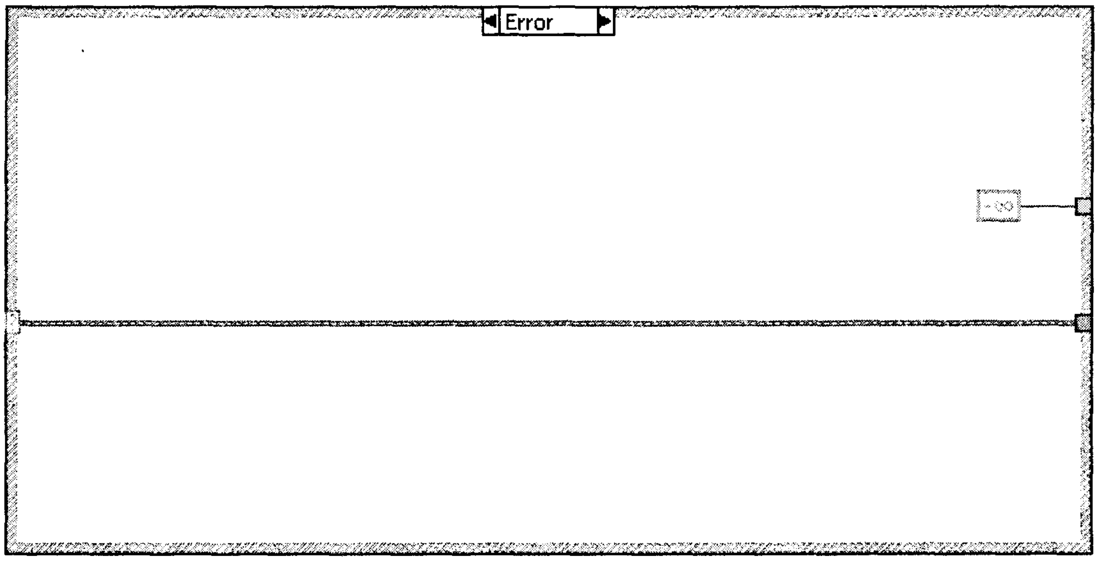
The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

gevent

2.14.3 Block Diagram







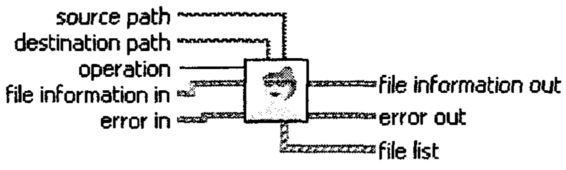
2.14.4 List of SubVIs

jLibrarian.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\jLibrarian.vi

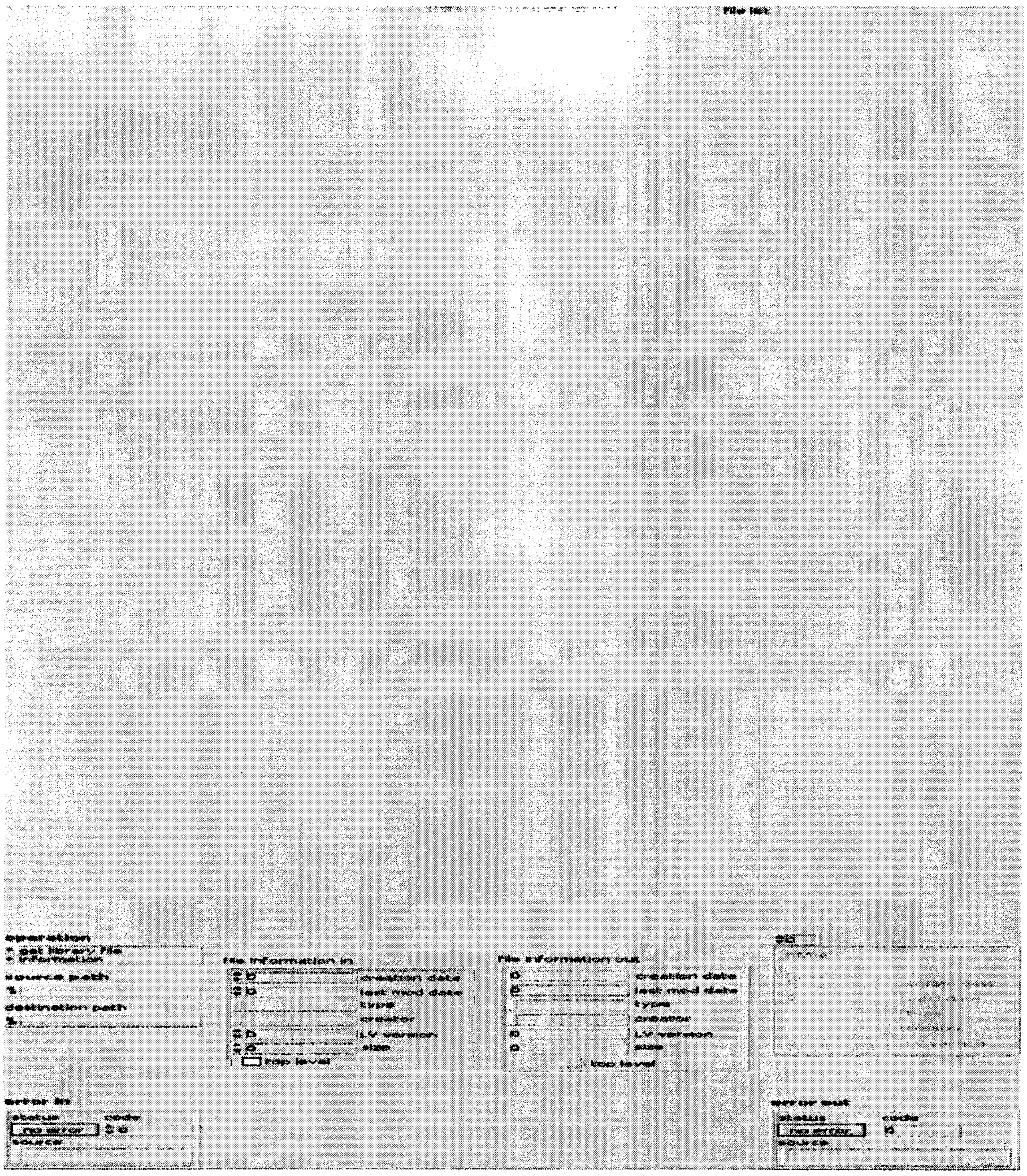
2.15 jLibrarian.vi

This VI contains the CIN that does all the real work on VI libraries.

2.15.1 Connector Pane



2.15.2 Front Panel



2.15.3 Controls and Indicators

file information in Information about a file. The creation date, mod date, and palette attributes apply to files in VI libraries. The type and creator apply to files that are not in VI libraries. The last modification date applies to directories. The size and LV version can not be changed.

creation date This is the date and time the file was created, expressed as seconds since January 1, 1904. This number can be converted to a string using the Get Date/Time String function in the Time & Dialog submenu. The creation date may be read or changed only for files in a VI library.

U32 **last mod date** This is the date and time when the file was last modified, expressed as seconds since January 1, 1904. This number can be converted to a string using the Get Date/Time String function. The last modification date may be read for all files but it may only be changed for files in a VI library.

abc **type** This is the type of the file, for example:
LVIN : VI
LVCC : custom control
LVAR : VI library
The type of a file in a VI library may not be changed.

abc **creator** This is the creator of the file, for example:
LabVIEW: LBVW
The creator of a file in a VI library may not be changed.

I32 **LV version** This is the version of LabVIEW in which the file was last saved. Use the I32 To LV Version VI to convert it to a LabVIEW version cluster and the LV Version To String VI to convert it to a string. The version may only be read (not changed) and only for files in a VI library.

I32 **size** Size of the uncompressed file. The size may be read but not changed.

TF **top level** TRUE if the file will be marked as a top-level VI in a VI library. This item may be read and changed only for files in a VI library.

error in Error encountered before entering this VI. If there was no error, this VI will execute. Otherwise, it will just pass the error value through.

TF **status** status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

I32 **code** code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

abc **source** source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

operation This is the operation to perform on the file or VI library at source path.

source path The path to the file or VI library to operate on.

destination path If the operation is move or copy, this is the destination for the new file.

file information out Information about a file. The palette information and version apply only to files in VI libraries. The creation date, mod date, and size apply to directories. For files not in VI libraries, the creation date, type, and creator are not valid on all platforms.

U32 **creation date** This is the date and time the file was created, expressed as seconds since January 1, 1904. This number can be converted to a string using the Get Date/Time String function in the Time & Dialog submenu. The creation date may be read or changed only for files in a VI library.

last mod date This is the date and time when the file was last modified, expressed as seconds since January 1, 1904. This number can be converted to a string using the Get Date/Time String function. The last modification date may be read for all files but it may only be changed for files in a VI library.

type This is the type of the file, for example:
LVIN : VI
LVCC : custom control
LVAR : VI library
The type of a file in a VI library may not be changed.

creator This is the creator of the file, for example:
LabVIEW: LBVW
The creator of a file in a VI library may not be changed.

LV version This is the version of LabVIEW in which the file was last saved. Use the I32 To LV Version VI to convert it to a LabVIEW version cluster and the LV Version To String VI to convert it to a string. The version may only be read (not changed) and only for files in a VI library.

size Size of the uncompressed file. The size may be read but not changed.

top level TRUE if the file is marked as a top-level VI in a VI library. This item may be read and changed only for files in a VI library.

error out If the status of error in is TRUE, this is a copy of error in. Otherwise, this is the first error encountered while executing this VI.

status status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

code code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

source source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

file list Information about each file in a VI library or directory.

file list Information about each file in a VI library or directory.

name Name of the file.

create date This is the date and time the file was created, expressed as seconds since January 1, 1904. This number can be converted to a string using the Get Date/Time String function in the Time & Dialog submenu. The creation date may be read or changed only for files in a VI library.

mod date This is the date and time when the file was last modified, expressed as seconds since January 1, 1904. This number can be converted to a string using the Get Date/Time String function. The last modification date may be read for all files but it may only be changed for files in a VI library.

type This is the type of the file, for example:
LVIN : VI
LVCC : custom control
LVAR : VI library
The type of a file in a VI library may not be changed.

creator This is the creator of the file, for example:
LabVIEW: LBVW
The creator of a file in a VI library may not be changed.

LV version This is the version of LabVIEW in which the file was last saved. Use the I32 To LV Version VI to convert it to a LabVIEW version cluster and the LV Version To String VI to convert it to a string. The version may only be read (not changed) and only for files in a VI library.

2.15.4 List of SubVIs

Librarian File Info In.ctl
C:\Program Files\National Instruments\LabVIEW 6\vi.lib\Utility\libraryn.llb\Librarian File Info In.ctl

Librarian File Info Out.ctl
C:\Program Files\National Instruments\LabVIEW 6\vi.lib\Utility\libraryn.llb\Librarian File Info Out.ctl

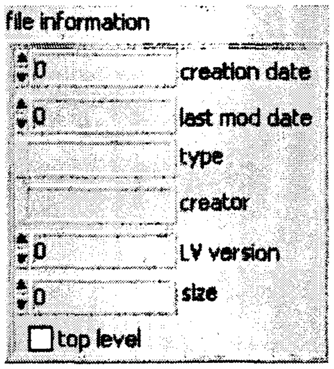
Librarian File List.ctl
C:\Program Files\National Instruments\LabVIEW 6\vi.lib\Utility\libraryn.llb\Librarian File List.ctl

2.16 Librarian File Info In.ctl

Information about a file. The creation date, mod date, and palette attributes apply to files in VI libraries. The type and creator apply to files that are not in VI libraries. The last modification date applies to directories. The size and LV version can not be changed.

2.16.1 Connector Pane

2.16.2 Front Panel



2.16.3 Controls and Indicators

file information Information about a file. The creation date, mod date, and palette attributes apply to files in VI libraries. The type and creator apply to files that are not in VI libraries. The last modification date applies to directories. The size and LV version can not be changed.

> U32 **creation date** This is the date and time the file was created, expressed as seconds since January 1, 1904. This number can be converted to a string using the Get Date/Time String function in the Time & Dialog submenu. The creation date may be read or changed only for files in a VI library.
>
> U32 **last mod date** This is the date and time when the file was last modified, expressed as seconds since January 1, 1904. This number can be converted to a string using the Get Date/Time String function. The last modification date may be read for all files but it may only be changed for files in a VI library.
>
> abc **type** This is the type of the file, for example:
> LVIN : VI
> LVCC : custom control
> LVAR : VI library
> The type of a file in a VI library may not be changed.
>
> abc **creator** This is the creator of the file, for example:
> LabVIEW: LBVW
> The creator of a file in a VI library may not be changed.
>
> I32 **LV version** This is the version of LabVIEW in which the file was last saved. Use the I32 To LV Version VI to convert it to a LabVIEW version cluster and the LV Version To String VI to convert it to a string. The version may only be read (not changed) and only for files in a VI library.
>
> I32 **size** Size of the uncompressed file. The size may be read but not changed.
>
> TF **top level** TRUE if the file will be marked as a top-level VI in a VI library. This item may be read and changed only for files in a VI library.

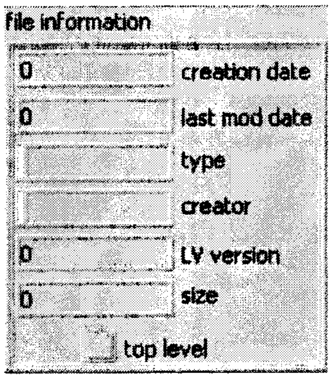
2.16.4 List of SubVIs

2.17 Librarian File Info Out.ctl

Information about a file. The palette information and version apply only to files in VI libraries. The creation date, mod date, and size apply to directories. For files not in VI libraries, the creation date, type, and creator are not valid on all platforms.

2.17.1 Connector Pane

2.17.2 Front Panel



2.17.3 Controls and Indicators

file information Information about a file. The palette information and version apply only to files in VI libraries. The creation date, mod date, and size apply to directories. For files not in VI libraries, the creation date, type, and creator are not valid on all platforms.

- **creation date** This is the date and time the file was created, expressed as seconds since January 1, 1904. This number can be converted to a string using the Get Date/Time String function in the Time & Dialog submenu. The creation date may be read or changed only for files in a VI library.

- **last mod date** This is the date and time when the file was last modified, expressed as seconds since January 1, 1904. This number can be converted to a string using the Get Date/Time String function. The last modification date may be read for all files but it may only be changed for files in a VI library.

- **type** This is the type of the file, for example:
  LVIN : VI
  LVCC : custom control
  LVAR : VI library
  The type of a file in a VI library may not be changed.

- **creator** This is the creator of the file, for example:
  LabVIEW: LBVW
  The creator of a file in a VI library may not be changed.

LV version This is the version of LabVIEW in which the file was last saved. Use the I32 To LV Version VI to convert it to a LabVIEW version cluster and the LV Version To String VI to convert it to a string. The version may only be read (not changed) and only for files in a VI library.

size Size of the uncompressed file. The size may be read but not changed.

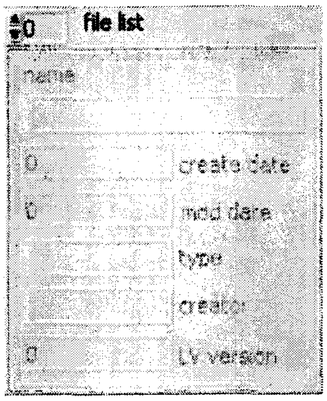
top level TRUE if the file is marked as a top-level VI in a VI library. This item may be read and changed only for files in a VI library.

2.17.4 List of SubVIs

2.18 Librarian File List.ctl

2.18.1 Connector Pane

2.18.2 Front Panel



2.18.3 Controls and Indicators

file list Information about each file in a VI library or directory.

file list Information about each file in a VI library or directory.

name Name of the file.

create date This is the date and time the file was created, expressed as seconds since January 1, 1904. This number can be converted to a string using the Get Date/Time String function in the Time & Dialog submenu. The creation date may be read or changed only for files in a VI library.

mod date This is the date and time when the file was last modified, expressed as seconds since January 1, 1904. This number can be converted to a string using the Get Date/Time String function. The last modification date may be read for all files but it may only be changed for files in a VI library.

type This is the type of the file, for example:
LVIN : VI
LVCC : custom control
LVAR : VI library
The type of a file in a VI library may not be changed.

creator This is the creator of the file, for example:
LabVIEW: LBVW
The creator of a file in a VI library may not be changed.

LV version This is the version of LabVIEW in which the file was last saved. Use the I32 To LV Version VI to convert it to a LabVIEW version cluster and the LV Version To String VI to convert it to a string. The version may only be read (not changed) and only for files in a VI library.

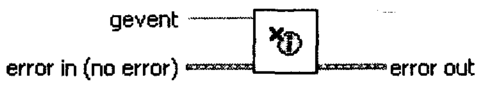
2.18.4 List of SubVIs

2.19 Destroy gevent.vi

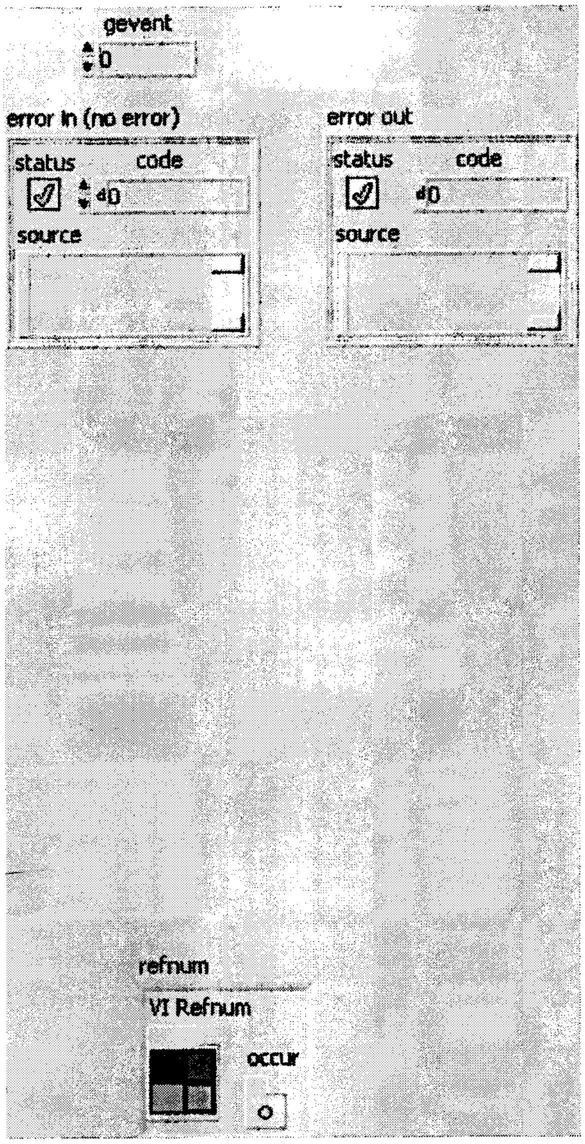
Destroys the specified gevent. Any wait on gevent vi that is currently waiting on this gevent will wake up immediately.

This VI will execute regardless of the state described by error in.

2.19.1 Connector Pane



2.19.2 Front Panel



2.19.3 Controls and Indicators

error in (no error) The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

status The status boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

code The code input identifies the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

source The source string describes the origin of the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

gevent

refnum

VI Refnum

occur

error out The error out cluster passes error or warning information out of a VI to be used by other VIs.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

status The status boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

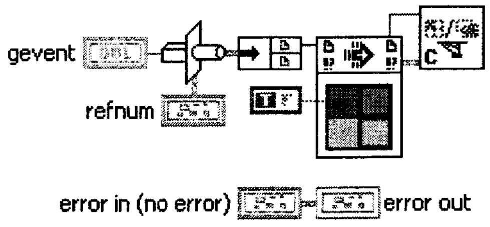
code The code input identifies the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

source The source string describes the origin of the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

2.19.4 Block Diagram



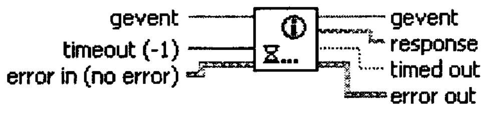
2.19.5 List of SubVIs

2.20 Wait on gevent.vi

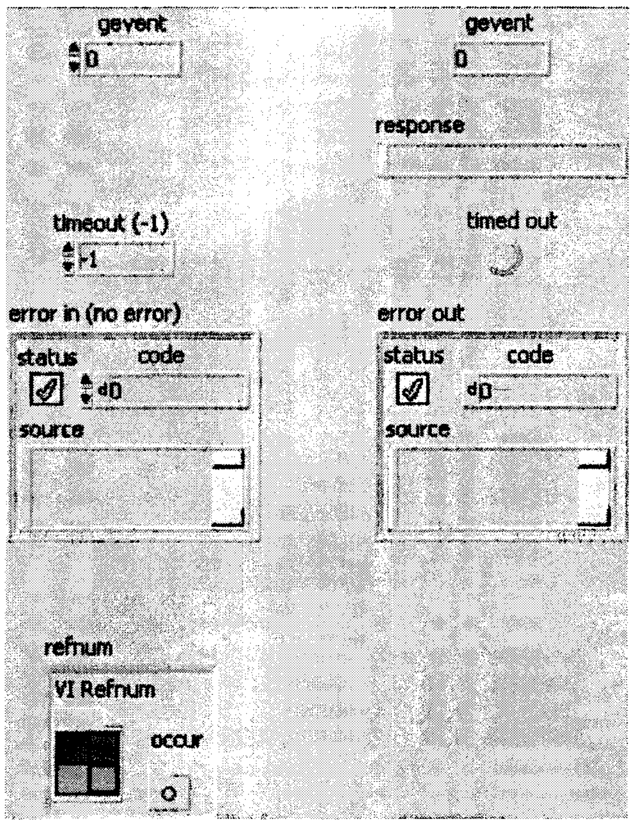
Waits for the specified gevent vi to become signaled. Timeout can be used to limit the time to wait for the event. Timeout defaults to waiting forever.

This VI will not work if error in specifies an error condition.

2.20.1 Connector Pane



2.20.2 Front Panel



2.20.3 Controls and Indicators

gevent

error in (no error) The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

- **status** The status boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

  The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

- **code** The code input identifies the error or warning.

  The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

- **source** The source string describes the origin of the error or warning.

  The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

timeout (-1) ms timeout specifies how many milliseconds the function waits for a notification to arrive.

refnum

- **VI Refnum**

- **occur**

error out The error out cluster passes error or warning information out of a VI to be used by other VIs.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

- **status** The status boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

  The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

- **code** The code input identifies the error or warning.

  The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

- **source** The source string describes the origin of the error or warning.

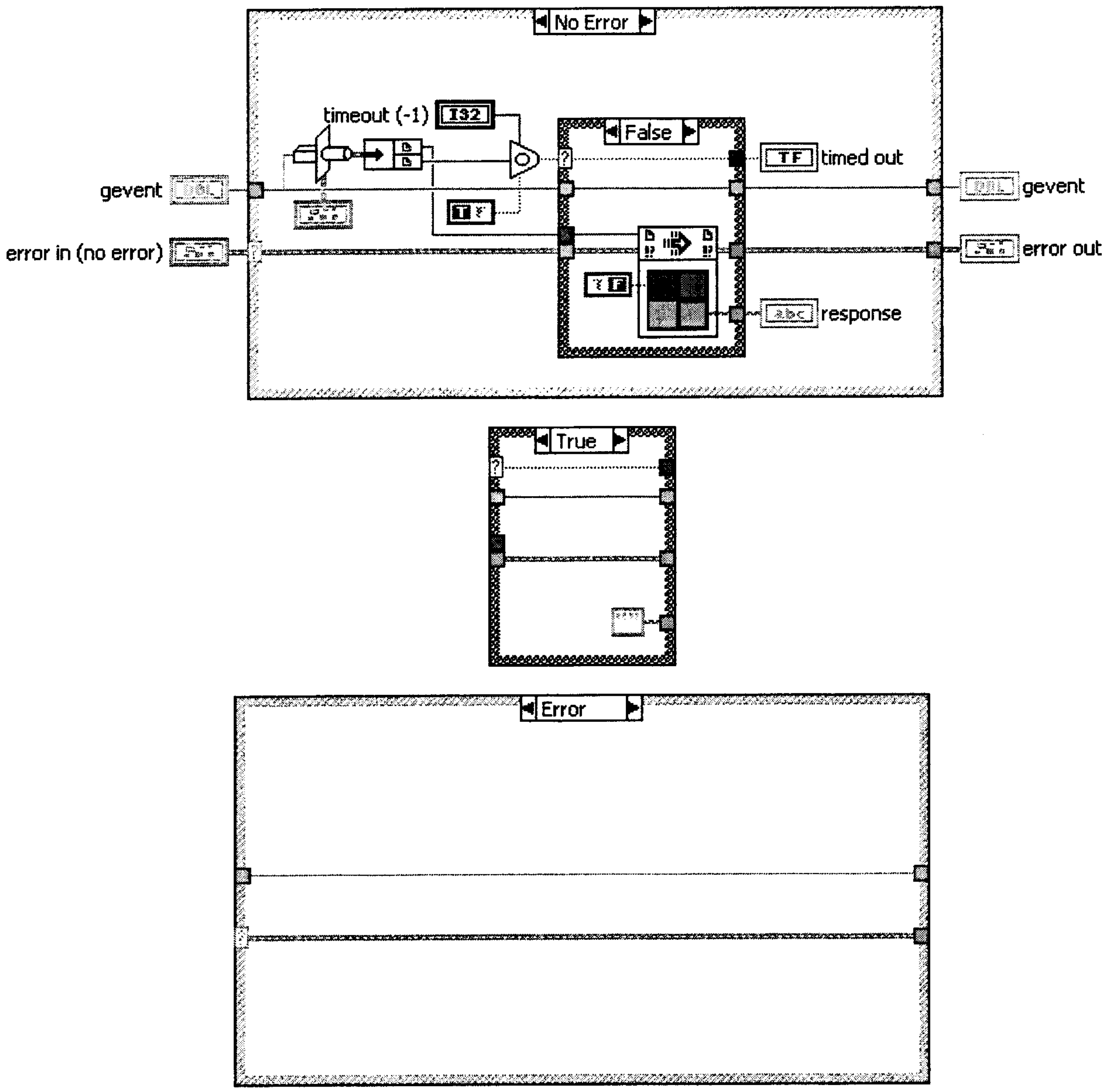
  The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

gevent

timed out timed out returns TRUE if no notification arrived before the function timed out or if an error occurred.

response notification contains the last notification sent to the notifier.

2.20.4 Block Diagram



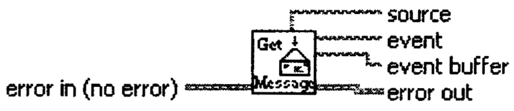
2.20.5 List of SubVIs

2.21 Get Message.vi

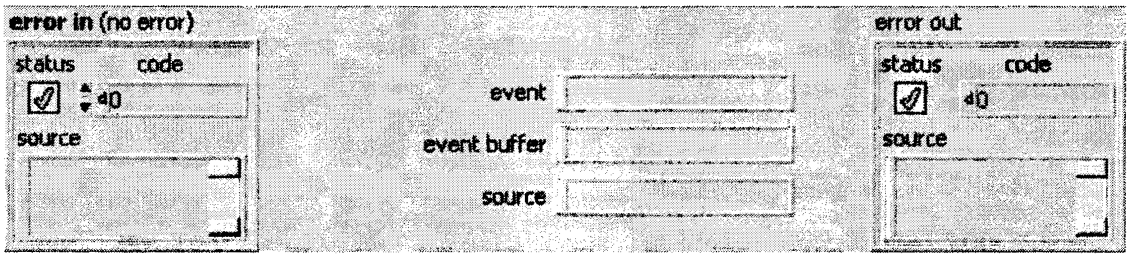
This VI retrieves a message from the private queue of the calling module. If there is an incoming error, this vi will NOT retrieve the message, but will propagate the error information to error out.

To receive messages, a module has to be registered by calling the Register Module.vi prior to sending it any messages.

2.21.1 Connector Pane



2.21.2 Front Panel



2.21.3 Controls and Indicators

error in (no error) The error in cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

- **status** The status boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

  The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

- **code** The code input identifies the error or warning.

  The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

- **source** The source string describes the origin of the error or warning.

  The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

event String that describes a task or event.

event buffer Data needed to perform the task specified by the event string.

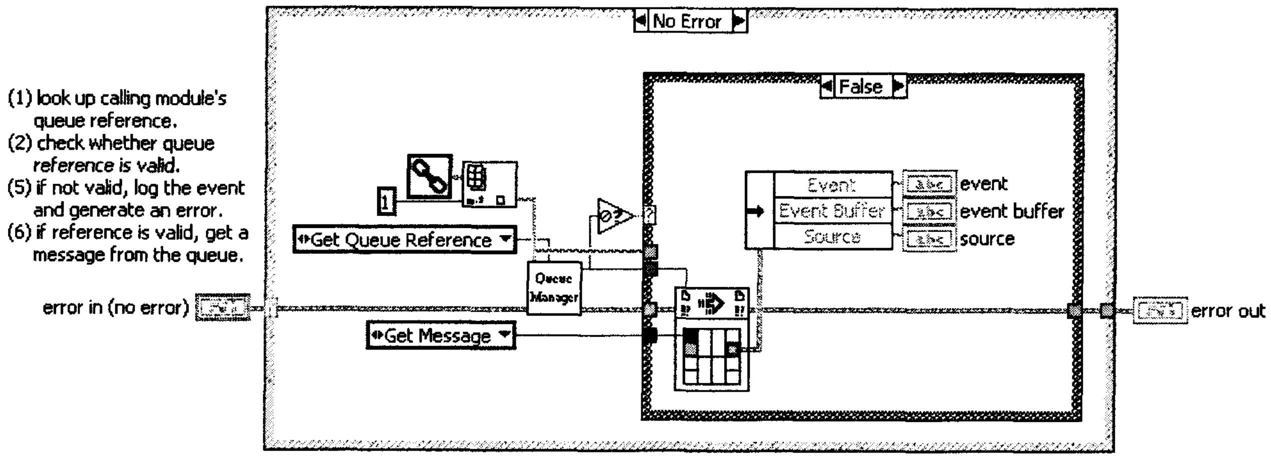
error out The error out cluster passes error or warning information out of a VI to be used by other VIs.

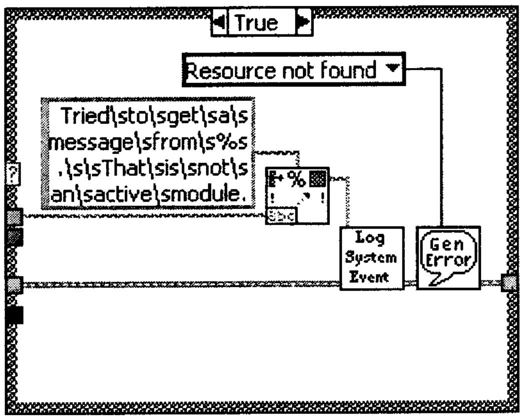
The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

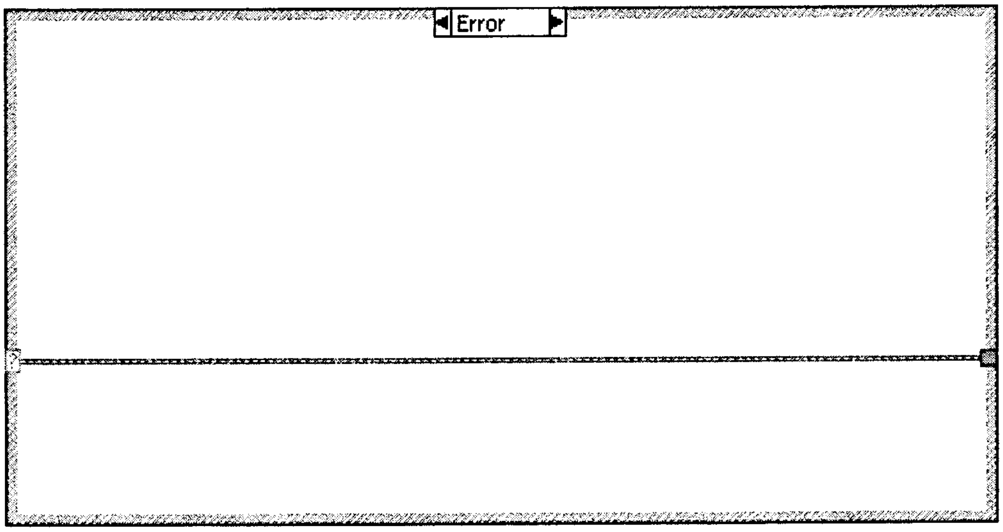
> **status** The status boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.
>
> The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.
>
> **code** The code input identifies the error or warning.
>
> The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.
>
> **source** The source string describes the origin of the error or warning.
>
> The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

source Data needed to perform the task specified by the event string.

2.21.4 Block Diagram


(1) look up calling module's queue reference.
(2) check whether queue reference is valid.
(5) if not valid, log the event and generate an error.
(6) if reference is valid, get a message from the queue.
error in (no error)
No Error
False
1
Get Queue Reference
Get Message
Queue Manager
Event
Event Buffer
Source
event
event buffer
source
error out


True
Resource not found
Tried\sto\sget\sa\s message\sfrom\s%s .\s\sThat\sis\snot\s an\sactive\smodule.
Log System Event
Gen Error



2.21.5 List of SubVIs

Queue Manager.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\Queue Manager.vi

log system event.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\log system event.vi

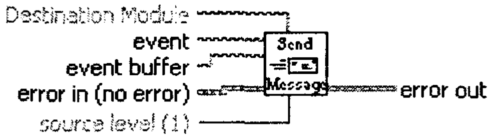
Generate Error.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\Generate Error.vi

2.22 Send Message.vi

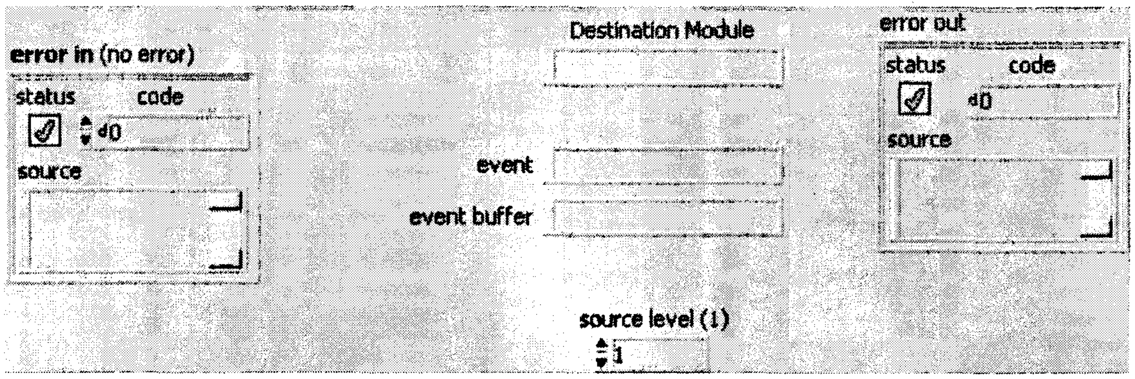
This VI adds a message to the private queue of the module specified by Destination Module. If Destination Module is not wired, the VI will send the message to the module it is running in. If there is an incoming error, this vi will NOT send the message, but will propagate the error information to error out.

To receive messages (whether from itself or other modules), a module has to be registered by calling the Register Module.vi prior to sending it any messages.

2.22.1 Connector Pane



2.22.2 Front Panel



2.22.3 Controls and Indicators

Destination Module Application module that will receive and handle the specified message.

event String that describes a task or event.

event buffer Data needed to perform the task specified by the event string.

error in (no error) The error in cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

> **status** The status boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.
>
> The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.
>
> **code** The code input identifies the error or warning.
>
> The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.
>
> **source** The source string describes the origin of the error or warning.
>
> The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

source level (1)

error out The error out cluster passes error or warning information out of a VI to be used by other VIs.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

status The status boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

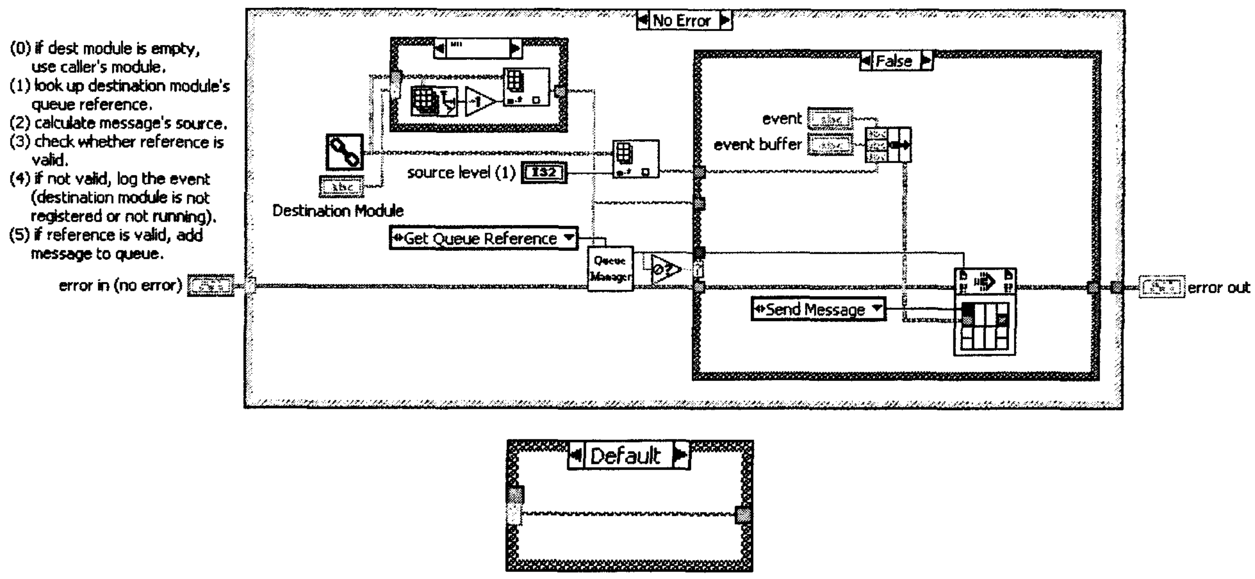
code The code input identifies the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

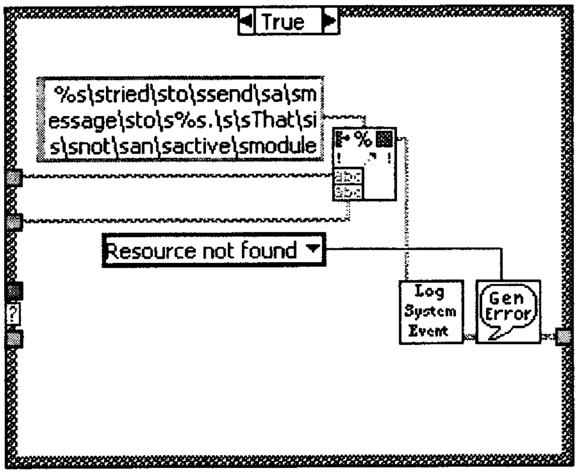
source The source string describes the origin of the error or warning.

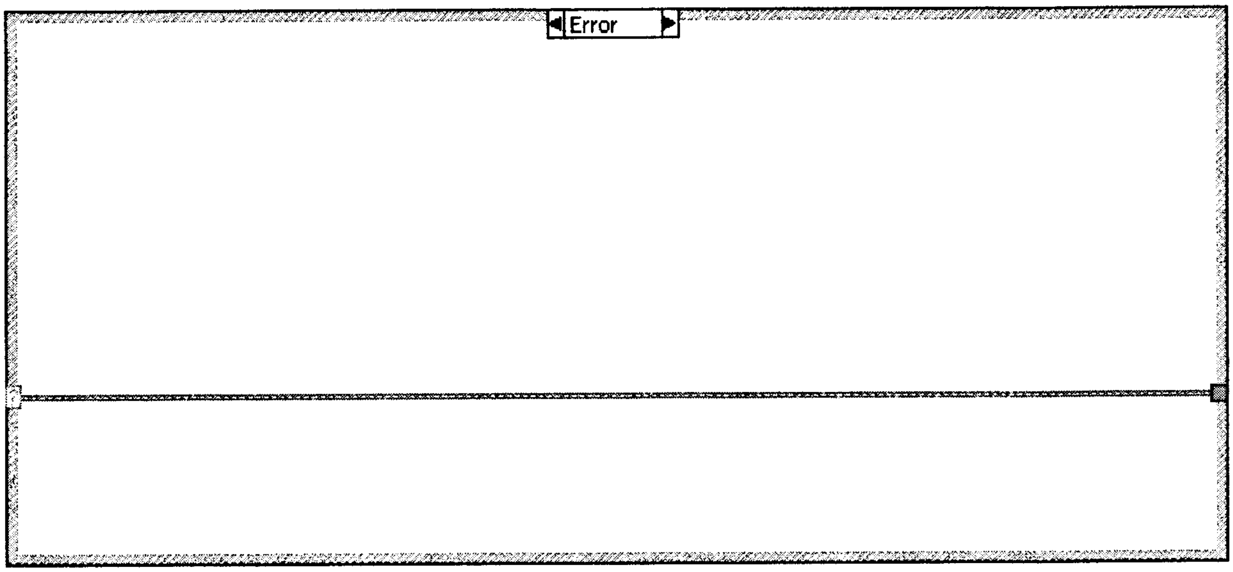
The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

2.22.4 Block Diagram



(0) if dest module is empty, use caller's module.
(1) look up destination module's queue reference.
(2) calculate message's source.
(3) check whether reference is valid.
(4) if not valid, log the event (destination module is not registered or not running).
(5) if reference is valid, add message to queue.


True
%s\stried\sto\ssend\sa\smessage\sto\s%s.\s\sThat\sis\snot\san\sactive\smodule
Resource not found
Log System Event
Gen Error



2.22.5 List of SubVIs

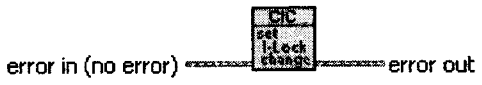
Queue Manager.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\Queue Manager.vi

log system event.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\log system event.vi

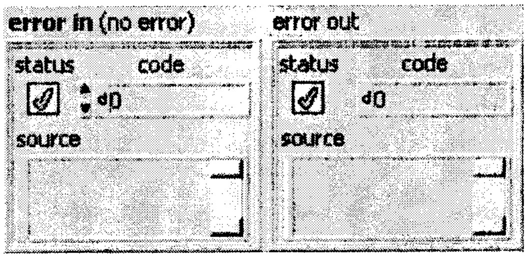
Generate Error.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\Generate Error.vi

2.23 CIC Set Interlock Setup Change.vi

2.23.1 Connector Pane



2.23.2 Front Panel



2.23.3 Controls and Indicators

error in (no error) The error in cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

> TF **status** The status boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.
>
> The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.
>
> I32 **code** The code input identifies the error or warning.
>
> The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.
>
> **source** The source string describes the origin of the error or warning.
>
> The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

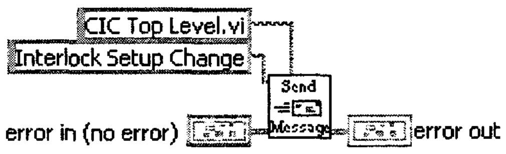
error out The error out cluster passes error or warning information out of a VI to be used by other VIs.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

> TF **status** The status boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.
>
> The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.
>
> I32 **code** The code input identifies the error or warning.
>
> The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.
>
> **source** The source string describes the origin of the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

2.23.4 Block Diagram



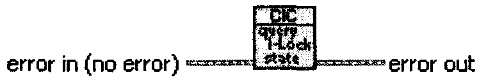
2.23.5 List of SubVIs

Send Message.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\Send Message.vi

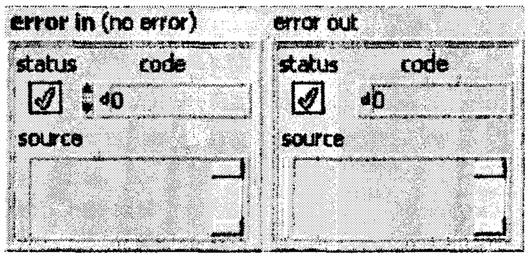
2.24 CIC Query Interlock State.vi

Sends the command to read and check the interlock status.

2.24.1 Connector Pane



2.24.2 Front Panel



2.24.3 Controls and Indicators

error in (no error) The error in cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

status The status boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

code The code input identifies the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

source The source string describes the origin of the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

error out The error out cluster passes error or warning information out of a VI to be used by other VIs.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

status The status boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

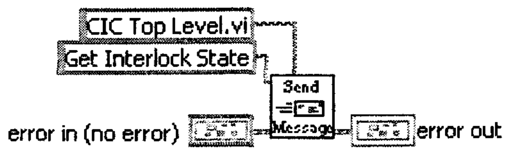
code The code input identifies the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

source The source string describes the origin of the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

2.24.4 Block Diagram

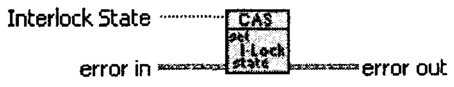


2.24.5 List of SubVIs

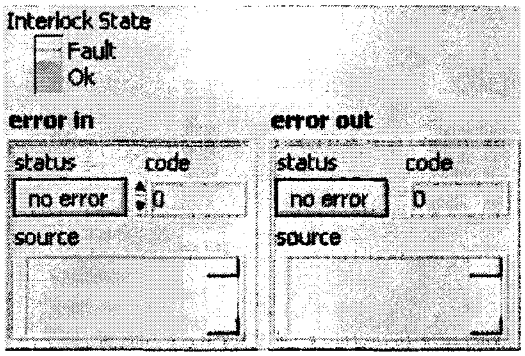
Send Message.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\Send Message.vi

2.25 CAS Set Interlock State.vi

2.25.1 Connector Pane



2.25.2 Front Panel



2.25.3 Controls and Indicators

error in error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

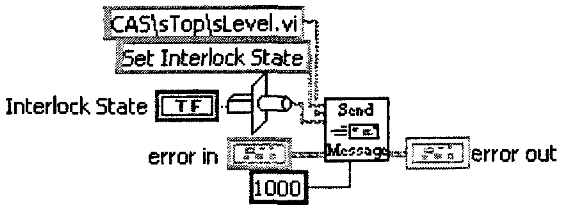
- **status** status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.
- **code** code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.
- **source** source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

Interlock State

error out error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

- **status** status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.
- **code** code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.
- **source** source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

2.25.4 Block Diagram

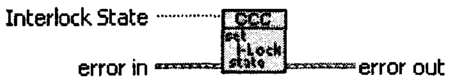


2.25.5 List of SubVIs

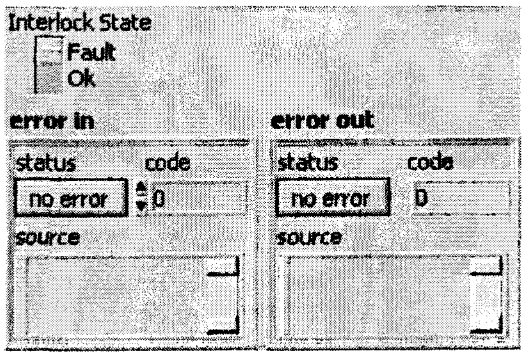
Send Message.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\Send Message.vi

2.26 CCC Set Interlock State.vi

2.26.1 Connector Pane



2.26.2 Front Panel



2.26.3 Controls and Indicators

error in error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

status status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

code code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

source source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

Interlock State

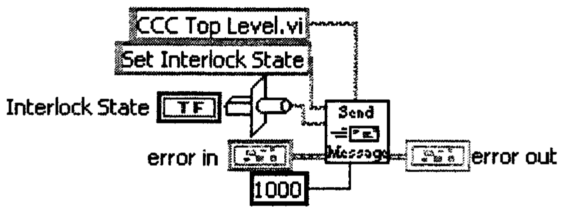
error out error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

status status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

code code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

source source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

2.26.4 Block Diagram

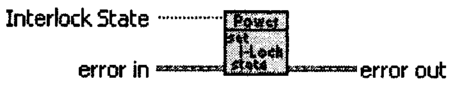


2.26.5 List of SubVIs

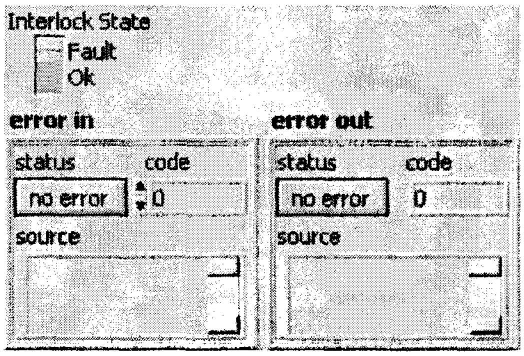
Send Message.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\Send Message.vi

2.27 PCS 01 001A Set Interlock State.vi

2.27.1 Connector Pane



2.27.2 Front Panel



2.27.3 Controls and Indicators

error in error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

- **status** status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

- **code** code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

- **source** source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

Interlock State

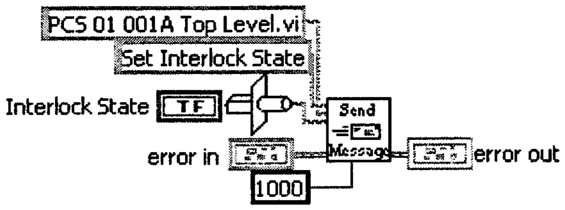
error out error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

- **status** status is TRUE if an error occurred, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

- **code** code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

- **source** source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

2.27.4 Block Diagram



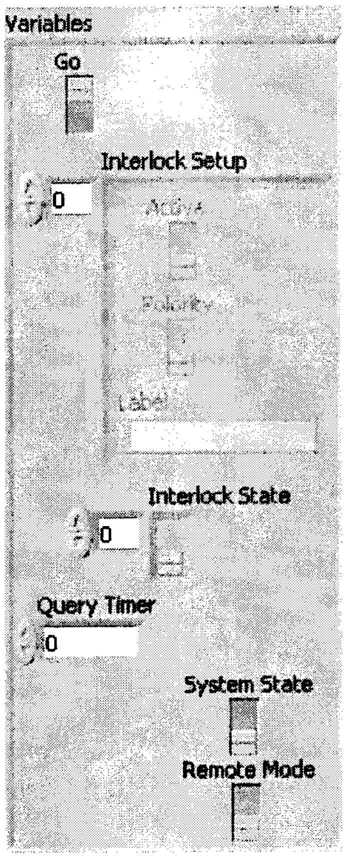
2.27.5 List of SubVIs

Send Message.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\Send Message.vi

2.28 CIC Variables.ctl

2.28.1 Connector Pane

2.28.2 Front Panel



2.28.3 Controls and Indicators

- Variables
  - Go
  - Interlock Setup
    - Channel 1
      - Active
      - Polarity
      - Label
  - Interlock State
    - Interlock State 1
  - Query Timer
  - System State

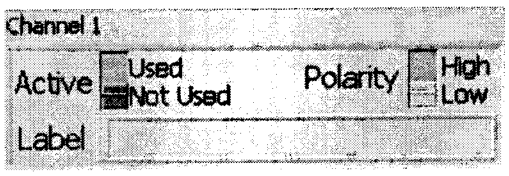
Remote Mode

2.28.4 List of SubVIs

2.29 CIC Channel Setup.ctl

2.29.1 Connector Pane

2.29.2 Front Panel



2.29.3 Controls and Indicators

Channel 1 Collectively, these controls define each interlock channel.

Active Determines if this channel is actively monitored as part of the Interlock System.

Polarity Determines the polarity state of the channel assigned to this interlock.

To successfully achieve an interlock, the channel must be active and the signal read by the DAQ Card must match the polarity assigned by this control.

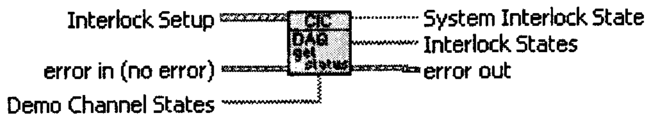
Label This is the arbitrary name of the interlock given by the user to define this channel.

2.29.4 List of SubVIs

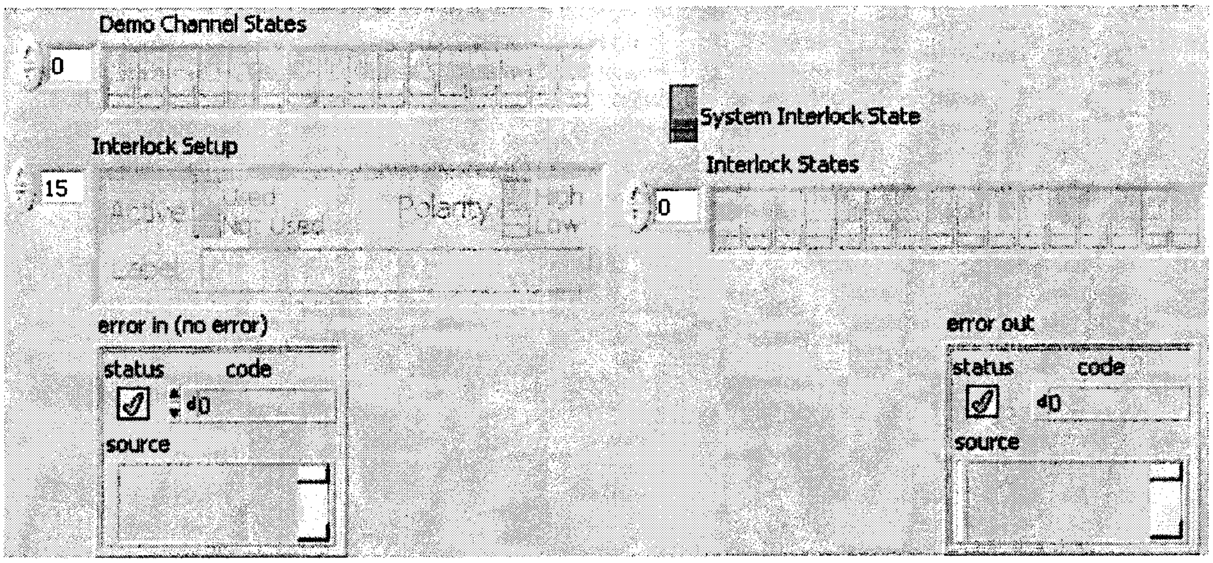
2.30 CIC DAQ Get Interlock Status.vi

This vi uses Daq DIO calls to read 16 channels of dio data and compares them to the interlock setup.

2.30.1 Connector Pane



2.30.2 Front Panel



2.30.3 Controls and Indicators

Demo Channel States

Boolean

Interlock Setup

Channel 1 Collectively, these controls define each interlock channel.

Active Determines if this channel is actively monitored as part of the Interlock System.

Polarity Determines the polarity state of the channel assigned to this interlock.

To successfully achieve an interlock, the channel must be active and the signal read by the DAQ Card must match the polarity assigned by this control.

Label This is the arbitrary name of the interlock given by the user to define this channel.

error in (no error) The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

status The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

code The **code** input identifies the error or warning.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

source The **source** string describes the origin of the error or warning.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

System Interlock State

Interlock States

Boolean

error out The **error out** cluster passes error or warning information out of a VI to be used by other VIs.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

status The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

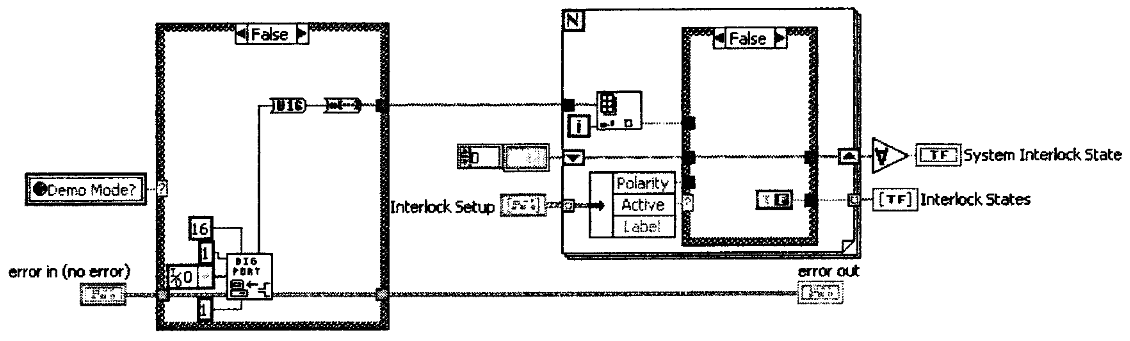
code The **code** input identifies the error or warning.

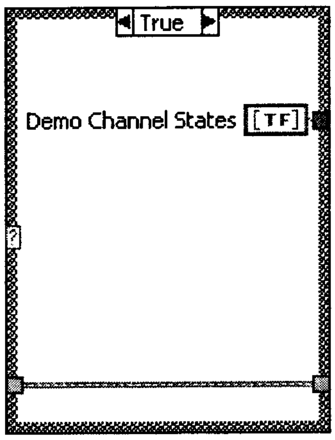
The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

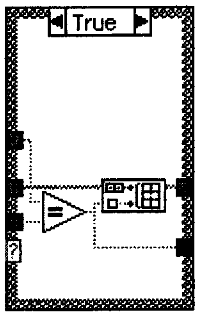
source The **source** string describes the origin of the error or warning.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

2.30.4 Block Diagram







2.30.5 List of SubVIs

CIC Channel Setup.ctl
C:\Controlatron\Development\Controlatron Interlock Control System\CIC Channel Setup.ctl

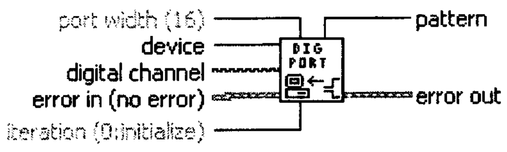
Read from Digital Port_With Error.vi
C:\Controlatron\Development\Controlatron Physical Drivers\DIO\Read from Digital Port_With Error.vi

Driver Demo Global.vi
C:\Controlatron\Development\Controlatron Physical Drivers\Driver Demo Global.vi

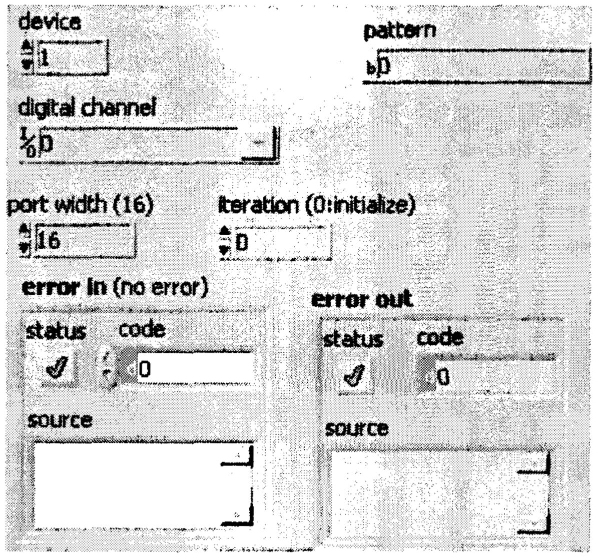
2.31 Read from Digital Port_With Error.vi

Reads a digital channel that you configure.

2.31.1 Connector Pane



2.31.2 Front Panel



2.31.3 Controls and Indicators

I16 **device device** is the device number you assigned to the DAQ device during configuration.

I/O **digital channel digital channel** is the channel name or port number that this VI configures.

I16 **port width (16) port width** is the total width or the number of lines of the port in bits.

I32 **iteration (0:initialize) iteration** can be used to optimize operation when you execute this VI in a loop.

error in (no error) The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

TF **status** The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

I32 **code** The **code** input identifies the error or warning.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

abc **source** The **source** string describes the origin of the error or warning.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

pattern **pattern** is the data the VI reads from the digital channel.

error out The **error out** cluster passes error or warning information out of a VI to be used by other VIs.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

status The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

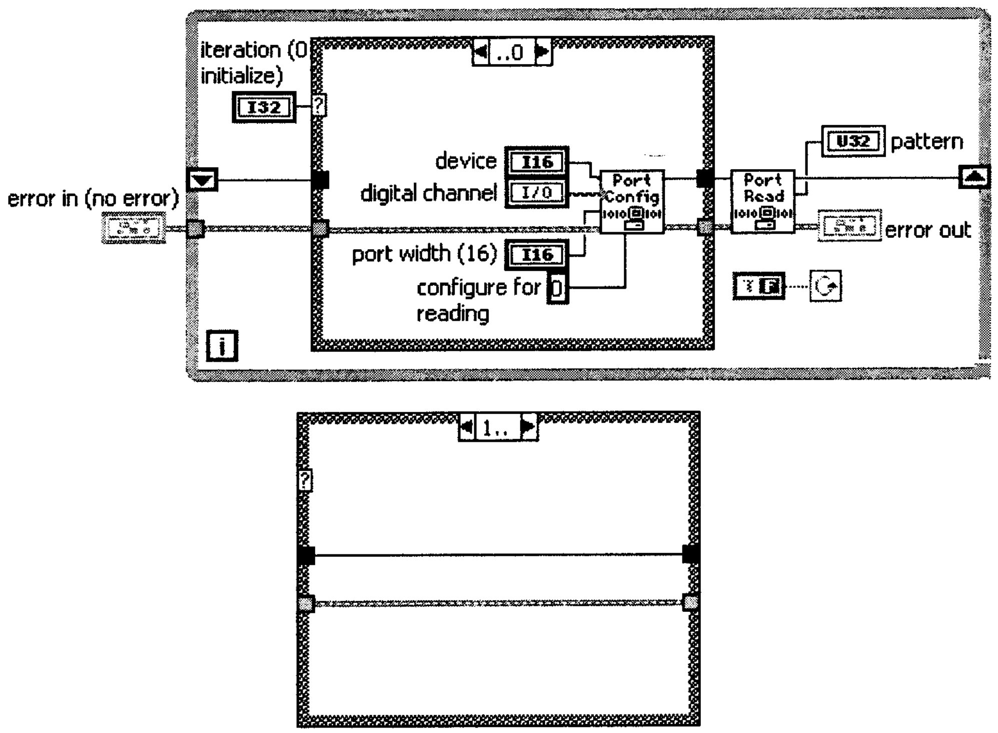
code The **code** input identifies the error or warning.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

source The **source** string describes the origin of the error or warning.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

2.31.4 Block Diagram



2.31.5 List of SubVIs

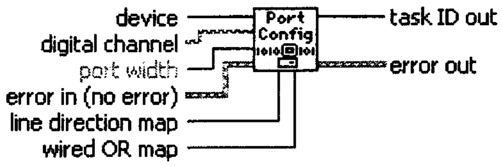
DIO Port Config.vi
C:\Program Files\National Instruments\LabVIEW 6\vi.lib\DAQ\ZADVD.LLB\DIO Port Config.vi

DIO Port Read.vi
C:\Program Files\National Instruments\LabVIEW 6\vi.lib\DAQ\ZADVD.LLB\DIO Port Read.vi

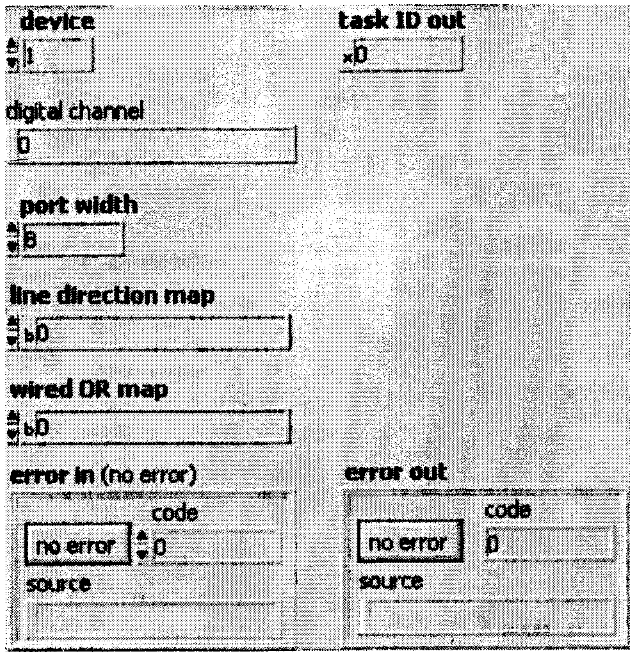
2.32 DIO Port Config.vi

Establishes a digital channel configuration. You can use the **task ID** that this VI returns only in digital port VIs.

2.32.1 Connector Pane



2.32.2 Front Panel



2.32.3 Controls and Indicators

device device is the device number you assigned to the DAQ device during configuration.

digital channel digital channel is the channel name or port number that this VI configures.

port width port width is the total width or the number of lines of the port in bits.

line direction map line direction map specifies the direction of each line in the port.

error in (no error) The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

status The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

code The **code** input identifies the error or warning.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

source The **source** string describes the origin of the error or warning.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

wired OR map wired OR map specifies whether each line in the output port is tri-state enabled.

task ID out task ID out uniquely identifies the digital group. Use this value as the task ID to refer to this group in subsequent digital port VIs.

error out The **error out** cluster passes error or warning information out of a VI to be used by other VIs.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

status The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

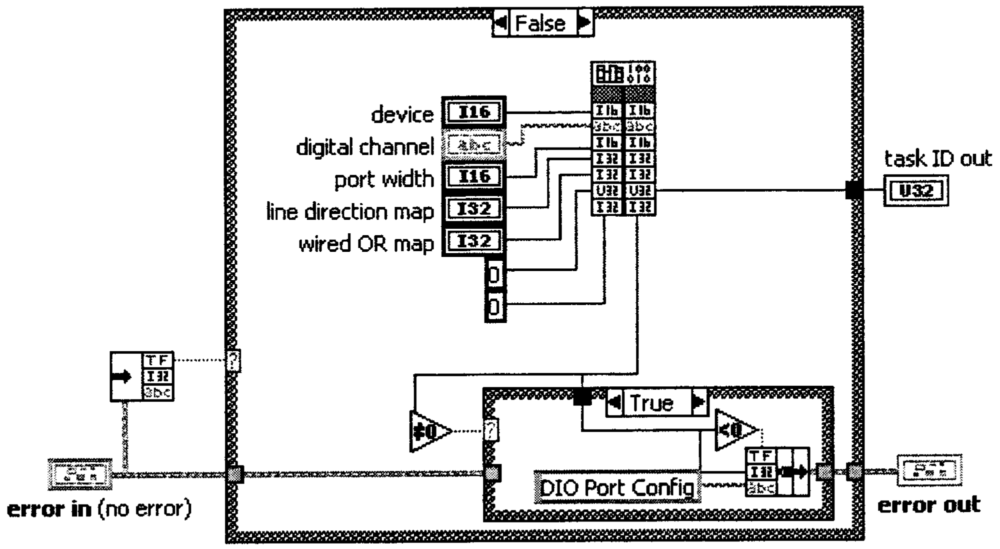
code The **code** input identifies the error or warning.

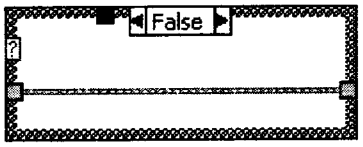
The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

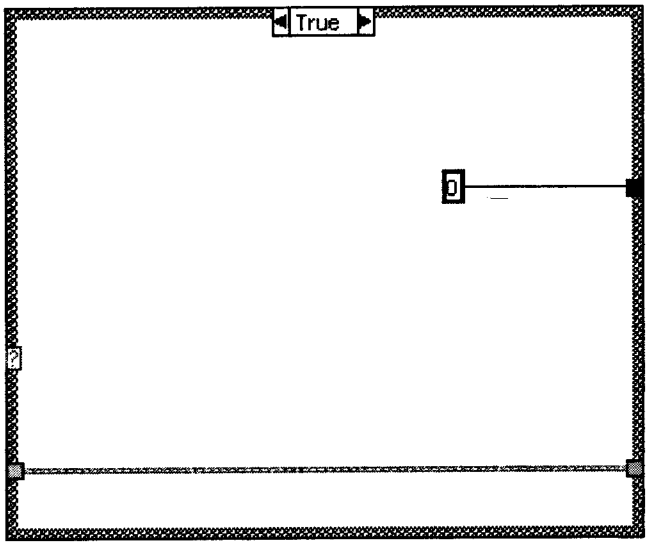
source The **source** string describes the origin of the error or warning.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

2.32.4 Block Diagram





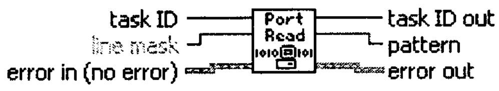


2.32.5 List of SubVIs

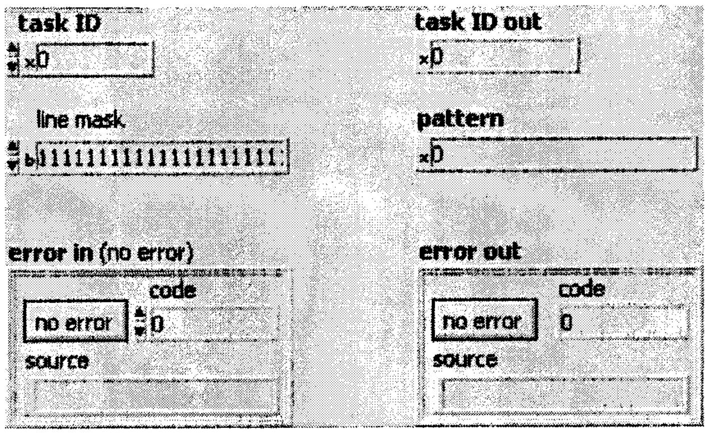
2.33 DIO Port Read.vi

Reads the digital channel identified by **task ID** and returns the pattern read in **pattern**.

2.33.1 Connector Pane



2.33.2 Front Panel



2.33.3 Controls and Indicators

U32 **task ID task ID** identifies the group and the I/O operation.

I32 **line mask line mask** determines which lines this VI reads.

error in (no error) The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

> TF **status** The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.
>
> The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.
>
> I32 **code** The **code** input identifies the error or warning.
>
> The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.
>
> abc **source** The **source** string describes the origin of the error or warning.
>
> The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

U32 **pattern pattern** -- If you use channel names where the channel name refers to a configured digital port or you do not use channel names, **pattern** is a bit pattern representing the state of the lines in the port. If the channel name is a configured line, **pattern** is 1 or 0.

task ID out task ID out uniquely identifies the digital group.

error out The **error out** cluster passes error or warning information out of a VI to be used by other VIs.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

- **status** The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

  The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

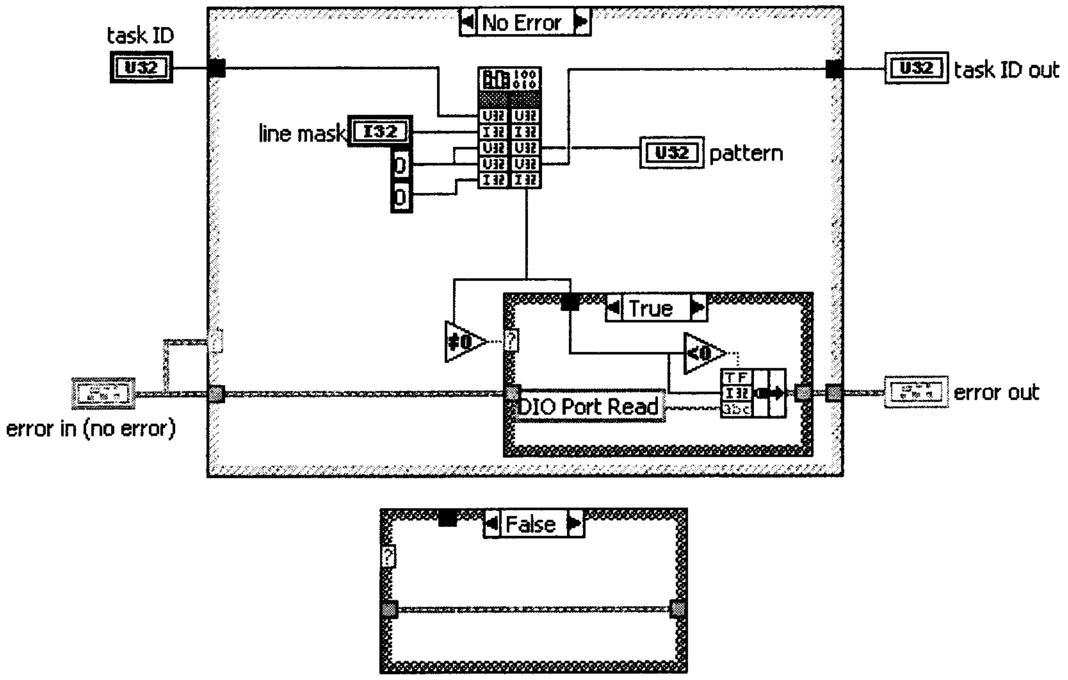
- **code** The **code** input identifies the error or warning.

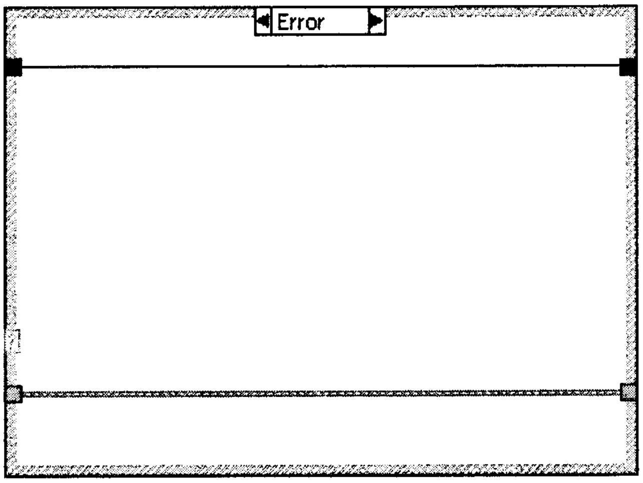
  The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

- **source** The **source** string describes the origin of the error or warning.

  The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

2.33.4 Block Diagram



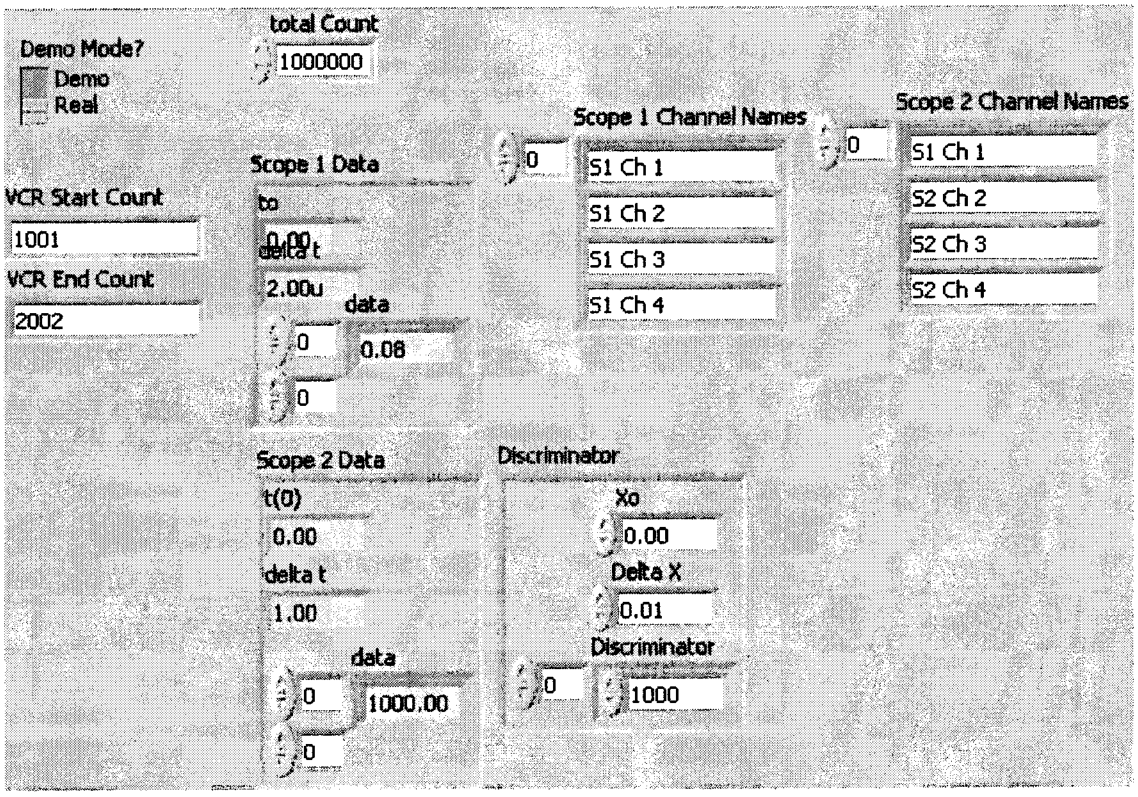


2.33.5 List of SubVIs

2.34 Driver Demo Global.vi

2.34.1 Connector Pane

2.34.2 Front Panel



2.34.3 Controls and Indicators

- [TF] **Demo Mode?**
- [abc] **VCR Start Count**
- [abc] **VCR End Count**
- [I32] **total Count**
- **Discriminator**
    - [DBL] **Xo**
    - [DBL] **Delta X**
    - [I32] **Discriminator**
        - [I32]
- [abc] **Scope 1 Channel Names**
    - [abc] **String**
- [abc] **Scope 2 Channel Names**
    - [abc] **String**
- **Scope 2 Data**
    - [EXT] **t(0)**
    - [EXT] **delta t**
    - [EXT] **data**
        - [EXT]
- **Scope 1 Data**
    - [DBL] **to**
    - [DBL] **delta t**
    - [DBL] **data**
        - [DBL]

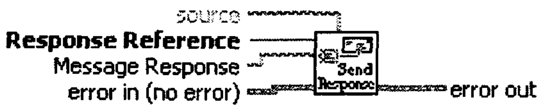
2.34.4 List of SubVIs

2.35 Respond To Message.vi

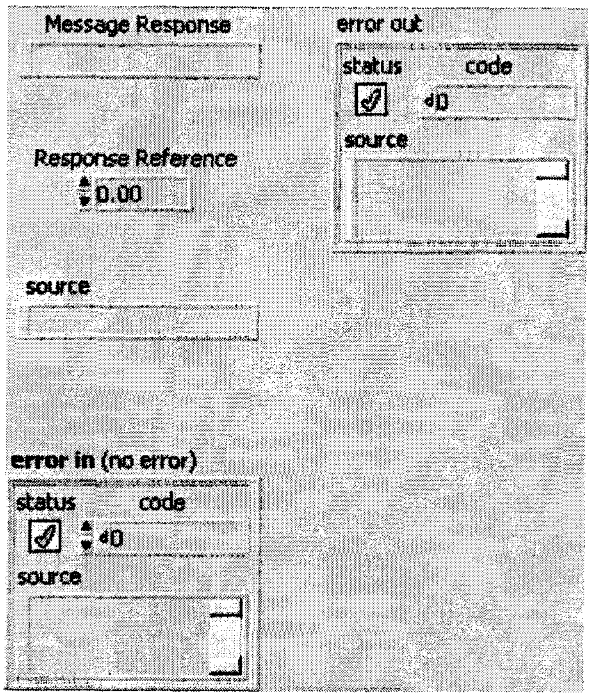
This VI sends a response to the originator of a response-requested message. You can get the Response Reference by calling the Get Response Reference.vi and passing it the event buffer of the response-requested message.

The response sent will include the error information passed in error in. Thus, this VI will send a response regardless of the error condition. Error out will always reflect the outcome of the response action.

2.35.1 Connector Pane



2.35.2 Front Panel



2.35.3 Controls and Indicators

Message Response Response to be sent as a result of a response-requested message.

Response Reference Reference that identifies the module that requested a response to the message. This reference must be used to send the response.

error in (no error) The error in cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

status The status boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

code The code input identifies the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

source The source string describes the origin of the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

source Response to be sent as a result of a response-requested message.

error out The error out cluster passes error or warning information out of a VI to be used by other VIs.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

status The status boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

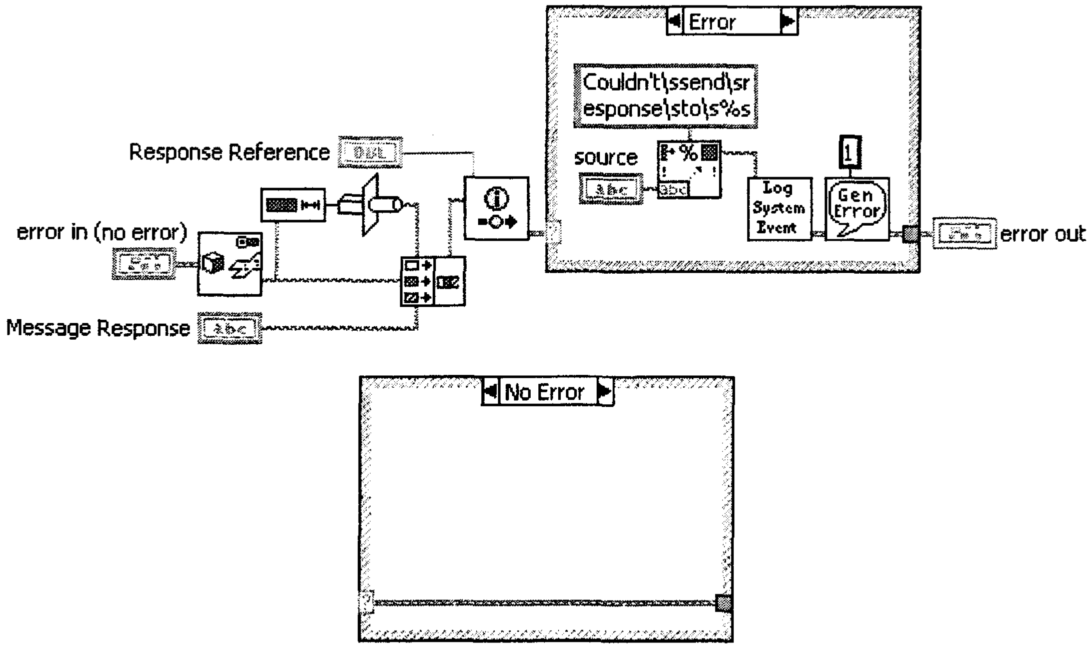
code The code input identifies the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

source The source string describes the origin of the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

2.35.4 Block Diagram



2.35.5 List of SubVIs

set gevent.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\set gevent.vi

Generate Error.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\Generate Error.vi

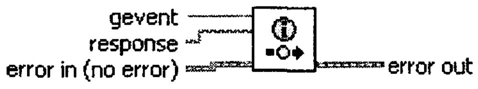
log system event.vi
C:\Controlatron\Development\msg v5.0.6\Msg5060.llb\log system event.vi

2.36 Set gevent.vi

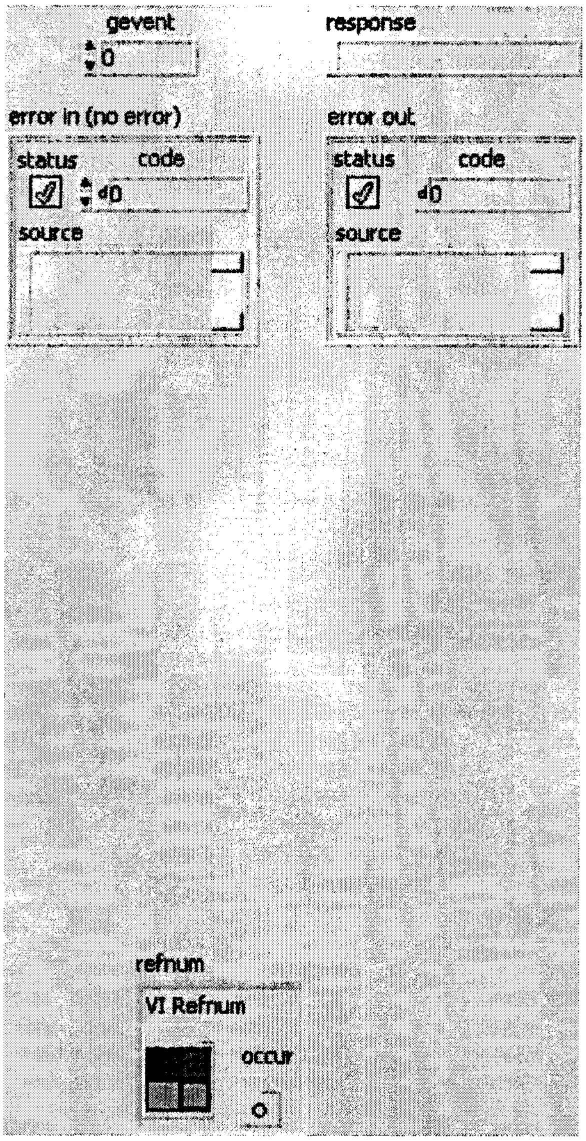
Set the specified gevent. Any wait on gevent vi that is currently waiting on this gevent will stop waiting and return the specified response.

This VI will not work if error in specifies an error condition.

2.36.1 Connector Pane



2.36.2 Front Panel



2.36.3 Controls and Indicators

error in (no error) The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

status The status boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

code The code input identifies the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

source The source string describes the origin of the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

gevent

refnum

VI Refnum

occur

response

error out The error out cluster passes error or warning information out of a VI to be used by other VIs.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

status The status boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

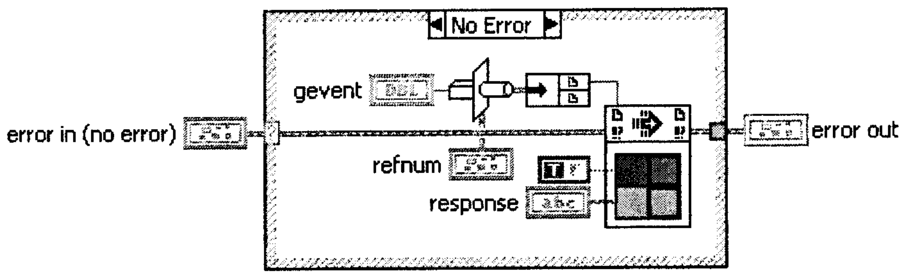
code The code input identifies the error or warning.

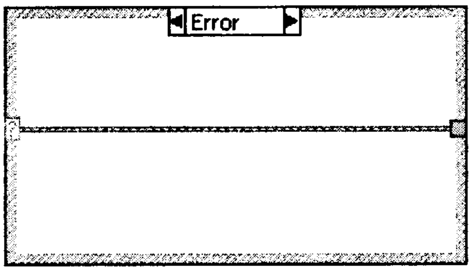
The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

source The source string describes the origin of the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

2.36.4 Block Diagram





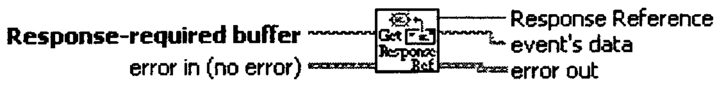
2.36.5 List of SubVIs

2.37 Get Response Reference.vi

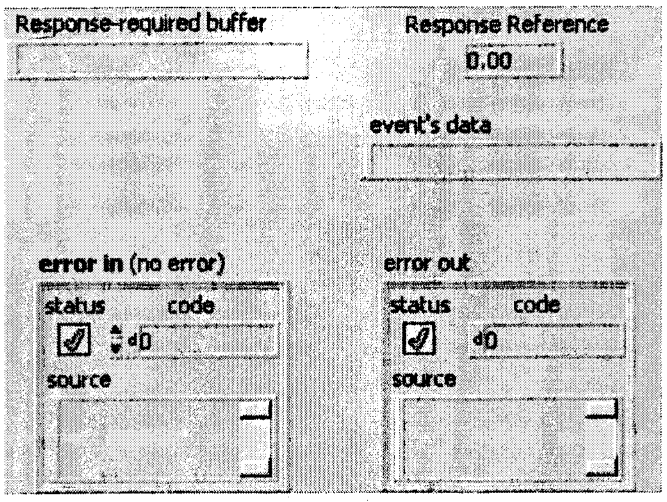
This VI parses out the data buffer of a response-required message. The data buffer consists of the message's data and the message originator's response reference. The module that handles the message must send a response to the module that sent the original message using the Respond To Message.vi.

This VI will execute regardless of the condition in the incoming error cluster.

2.37.1 Connector Pane



2.37.2 Front Panel



2.37.3 Controls and Indicators

Response-required buffer Event buffer for a response-required message. A message that requires a response sends the response reference included in its event buffer element.

error in (no error) The error in cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

status The status boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

code The code input identifies the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

source The source string describes the origin of the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

Response Reference Reference that identifies the module that is requesting a response to the message. This reference must be used to send a response to the message originator.

event's data Data needed to perform the task specified by the message's event.

error out The error out cluster passes error or warning information out of a VI to be used by other VIs.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

status The status boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

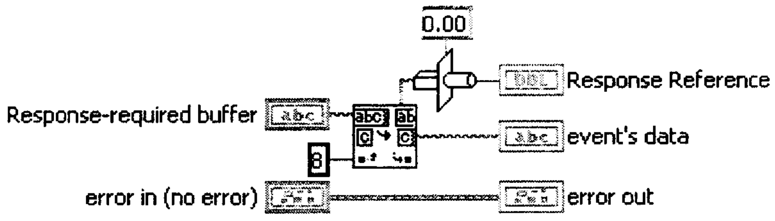
code The code input identifies the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

source The source string describes the origin of the error or warning.

The pop-up option Explain Error (or Explain Warning) gives more information about the error displayed.

2.37.4 Block Diagram

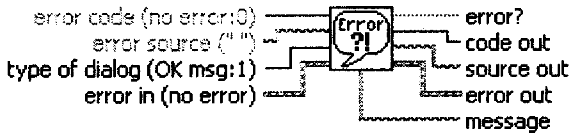


2.37.5 List of SubVIs

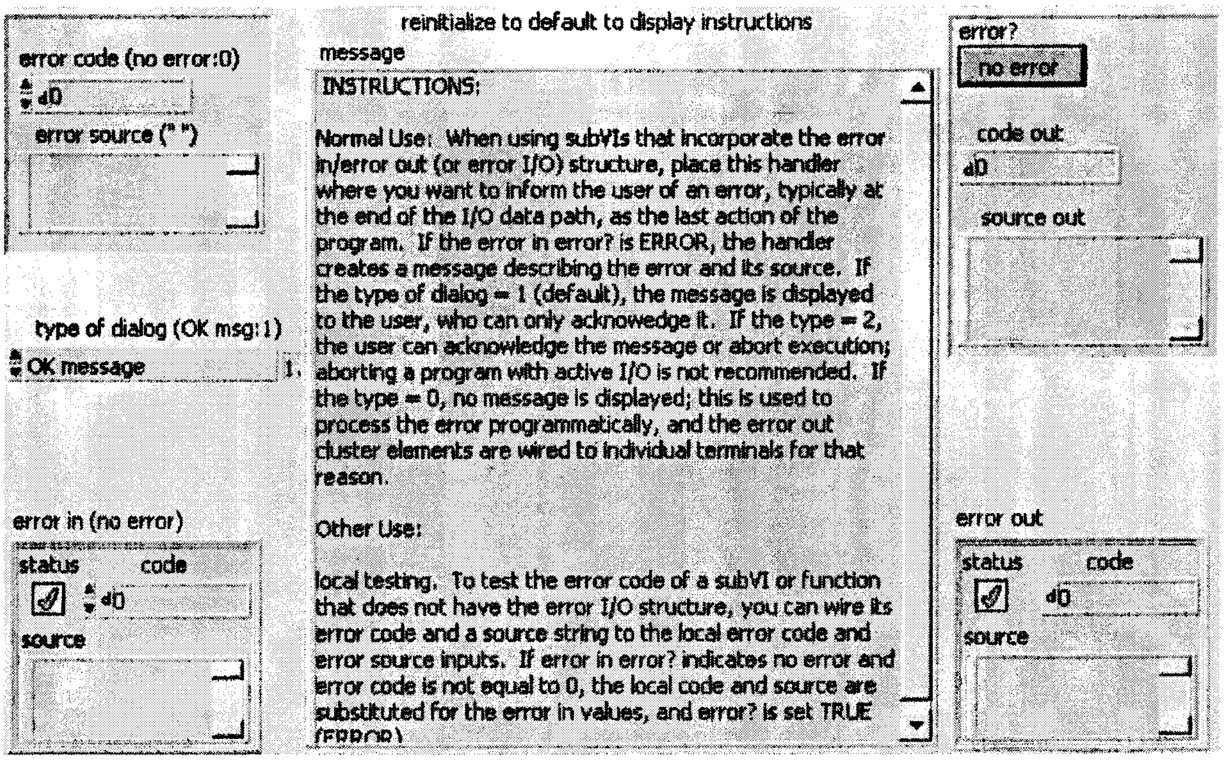
2.38 Simple Error Handler.vi

Determines whether an error occurred. It contains a database of error codes and descriptions, from which it creates a description of the error and optionally displays a dialog box.

2.38.1 Connector Pane



2.38.2 Front Panel



INSTRUCTIONS:

Normal Use: When using subVIs that incorporate the error in/error out (or error I/O) structure, place this handler where you want to inform the user of an error, typically at the end of the I/O data path, as the last action of the program. If the error in error? is ERROR, the handler creates a message describing the error and its source. If the type of dialog = 1 (default), the message is displayed to the user, who can only acknowedge it. If the type = 2, the user can acknowledge the message or abort execution; aborting a program with active I/O is not recommended. If the type = 0, no message is displayed; this is used to process the error programmatically, and the error out cluster elements are wired to individual terminals for that reason.

Other Use:

local testing. To test the error code of a subVI or function that does not have the error I/O structure, you can wire its error code and a source string to the local error code and error source inputs. If error in error? indicates no error and error code is not equal to 0, the local code and source are substituted for the error in values, and error? is set TRUE (ERROR)

2.38.3 Controls and Indicators

type of dialog (OK msg:1) type of dialog determines what type of dialog box should be displayed, if any. Regardless of its value, the VI outputs error information and a **message** describing the error.

error source (" ") error source is an optional string that you can use to describe the source of error code.

error code (no error:0) error code is a numeric error code.

error in (no error) The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

> **status** The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.
>
> The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.
>
> **code** The **code** numeric identifies the error or warning.
>
> The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.
>
> **source** The **source** string describes the origin of the error or warning.
>
> The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

message message describes the error code that occurred, the source of the error, and a description of the error.

code out code out is the error code indicated by the **error in** or **error code.**

source out source out indicates the source of the error.

error? error? indicates if the VI found an error. If this VI finds an error, it sets the parameters in the error cluster.

error out The **error out** cluster passes error or warning information out of a VI to be used by other VIs.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

> **status** The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.
>
> The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

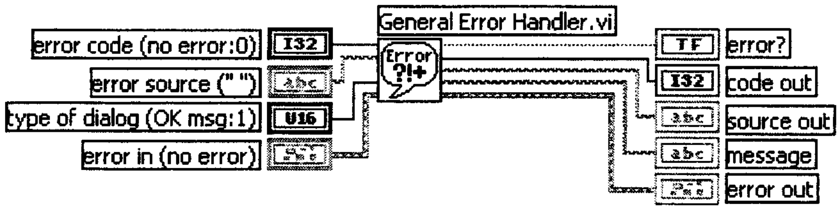
I32 **code** The **code** numeric identifies the error or warning.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

abc **source** The **source** string describes the origin of the error or warning.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

2.38.4 Block Diagram

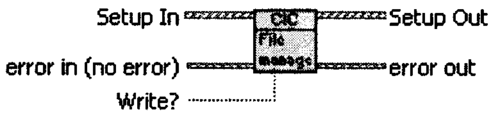


2.38.5 List of SubVIs

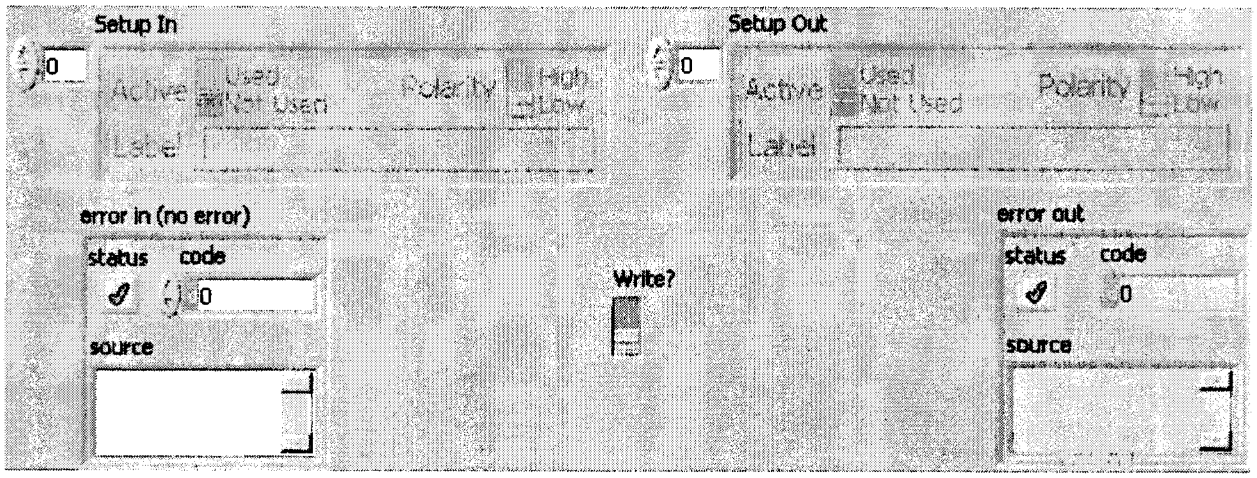
General Error Handler.vi
C:\Program Files\National Instruments\LabVIEW 6\vi.lib\Utility\error.llb\General Error Handler.vi

2.39 CIC File Management.vi

2.39.1 Connector Pane



2.39.2 Front Panel



2.39.3 Controls and Indicators

error in (no error) The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

- **status** The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

  The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

- **code** The **code** input identifies the error or warning.

  The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

- **source** The **source** string describes the origin of the error or warning.

  The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

Setup In

- **Channel 1** Collectively, these controls define each interlock channel.

  - **Active** Determines if this channel is actively monitored as part of the Interlock System.

  - **Polarity** Determines the polarity state of the channel assigned to this interlock.

    To successfully achieve an interlock, the channel must be active and the signal read by the DAQ Card must match the polarity assigned by this control.

  - **Label** This is the arbitrary name of the interlock given by the user to define this channel.

Write?

error out The **error out** cluster passes error or warning information out of a VI to be used by other VIs.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

- **status** The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

  The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

- **code** The **code** input identifies the error or warning.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

source The **source** string describes the origin of the error or warning.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

Setup Out

Channel 1 Collectively, these controls define each interlock channel.

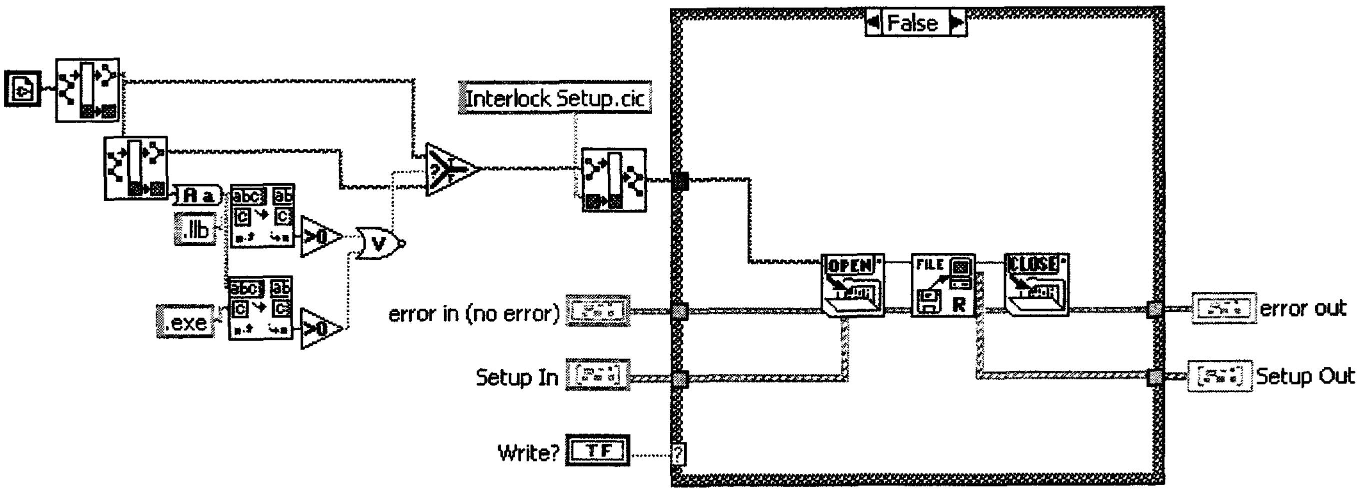
Active Determines if this channel is actively monitored as part of the Interlock System.

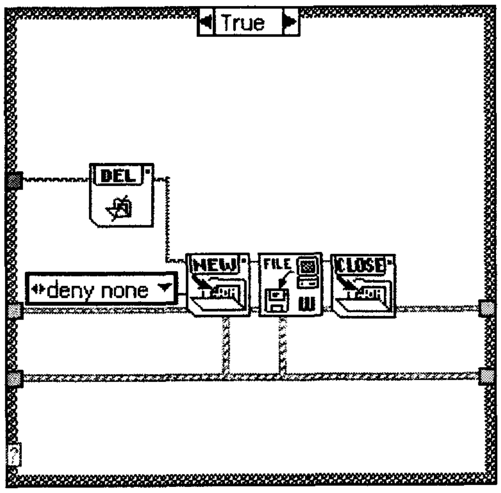
Polarity Determines the polarity state of the channel assigned to this interlock.

To successfully achieve an interlock, the channel must be active and the signal read by the DAQ Card must match the polarity assigned by this control.

Label This is the arbitrary name of the interlock given by the user to define this channel.

2.39.4 Block Diagram





2.39.5 List of SubVIs

CIC Channel Setup.ctl
C:\Controlatron\Development\Controlatron Interlock Control System\CIC Channel Setup.ctl

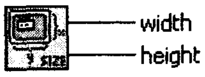
2.40 Get Screen Size.vi

Gets the size of the screen in pixels.

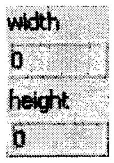
The LabVIEW Technical Resource newsletter (LTR) originally distributed this VI.
The LTR is a quarterly publication providing technical information to LabVIEW system developers.

For subscription information, contact:
LTR Publishing, Inc.
6060 N. Central Expressway, Suite 502
Dallas, TX 75206
(214) 706-0587
FAX (214) 706-0506
E-mail: subscribe@ltrpub.com
www.ltrpub.com

2.40.1 Connector Pane



2.40.2 Front Panel

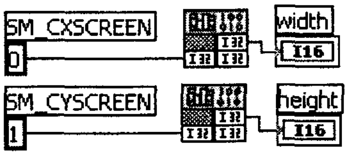


2.40.3 Controls and Indicators

I16 **width** Width of the screen in pixels.

I16 **height** Height of the screen in pixels.

2.40.4 Block Diagram



2.40.5 List of SubVIs

Distribution:

6		Primecore Systems, Inc. Attn: Robert Hertrich (2) Keith Barrett (2) Archives (2) 8421 Osuna Rd. Suite C-1 Albuquerque, NM 87111
1	MS-0515	R. W. Howe, 2561
1	0516	D. L. Wallace, 2564
1	0516	J. T. Crow, 2564
1	0516	G. G. Gonzales, 2564
5	1183	W. P. Noel, 15425
1	1183	M. L. Martinez, 15425
1	9018	Central Technical Files, 8945-1
2	0899	Technical Library, 9616
1	0612	Review and Approval Desk, 9612 For DOE/OSTI